# Posicionament angular de dispositius en 3 plans ortogonals mitjançant processat de senyals de velocitat angular

Oriol Carro i Vidallet

gener de 2010

# Índex

# Índex de taules

# Índex de figures

# Capítol 1

# Introducció

## 1.1 Objectius

L'objectiu principal és implementar un sistema que determini en cada instant quin ha estat el moviment angular d'un objecte en cadascun dels 3 plans que conformen l'espai (3 dimensions). És a dir, s'ha de poder determinar en tot moment quina és la ubicació (punt exacte) que ocupa en l'espai el dispositiu mòbil estudiat a partir de la velocitat angular amb que aquest es mou en cada pla.

A part de l'objectiu principal, hi ha d'altres objectius més específics que ens ajudaran a assolir la finalitat del projecte, és a dir, tasques que cal realitzar per tal de poder arribar a una solució del problema.

- Desenvolupar aplicacions emprant el llenguatge C, tenint en compte que la quantitat de memòria que disposem està limitada pels components escollits i que el seu rendiment ha de ésser el més elevat possible.
- Programar el microcontrolador MCF5213 mitjançant l'eina de desenvolupament Code Warrior 7.1.
- Conèixer els diferents mòduls que composen el xip en qüestió, distingint quins ens poden ésser útils en cada moment, i saber-los utilitzar de forma correcta.

Cal tenir en compte que aquests són els objectius per aquest treball, però que no s'ha de perdre de vista, que aquest s'engloba dins d'un projecte de major envergadura.

## 1.2 Punt de partida

El punt de partida d'aquest projecte és un altre treball final de carrera [1] en què s'estudiava quina era la millor forma de mesurar la magnitud i el sentit de la rotació angular que té un objecte en l'espai i que aquesta fos viable d'implementar. Així doncs, de les conclusions d'aquest, en podem extreure que escollir un giroscopi amb tecnologia MEMS és una bona solució. Tot i no ésser tant precisos com els giroscopis òptics, es va considerar que els resultats obtinguts eren prou satisfactoris, permetent així, descartar la utilització d'un giroscopi òptic, el cost del qual és molt més elevat. Més concretament, el giroscopi escollit fou el ADXRS610. En el capítol de components se'n fa una breu descripció.

A part del dispositiu (giroscopi) que cal utilitzar per tal de poder obtenir les velocitats angulars amb que un objecte es mou per l'espai, també disposem, en el mateix projecte citat anteriorment, d'un estudi de l'algorisme que ens permetrà a partir d'aquestes dades, calcular quin és el desplaçament produït en cadascun dels plans.

Així doncs, l'organigrama resum que descriu l'algorisme a implementar és el següent:

[1] Aplicació d'estratègies de processament de senyals variants amb la velocitat angular del dispositiu sensor ADXRS610 de Robert Rocamora Graell i dirigit per Francisco Clariá Sancho

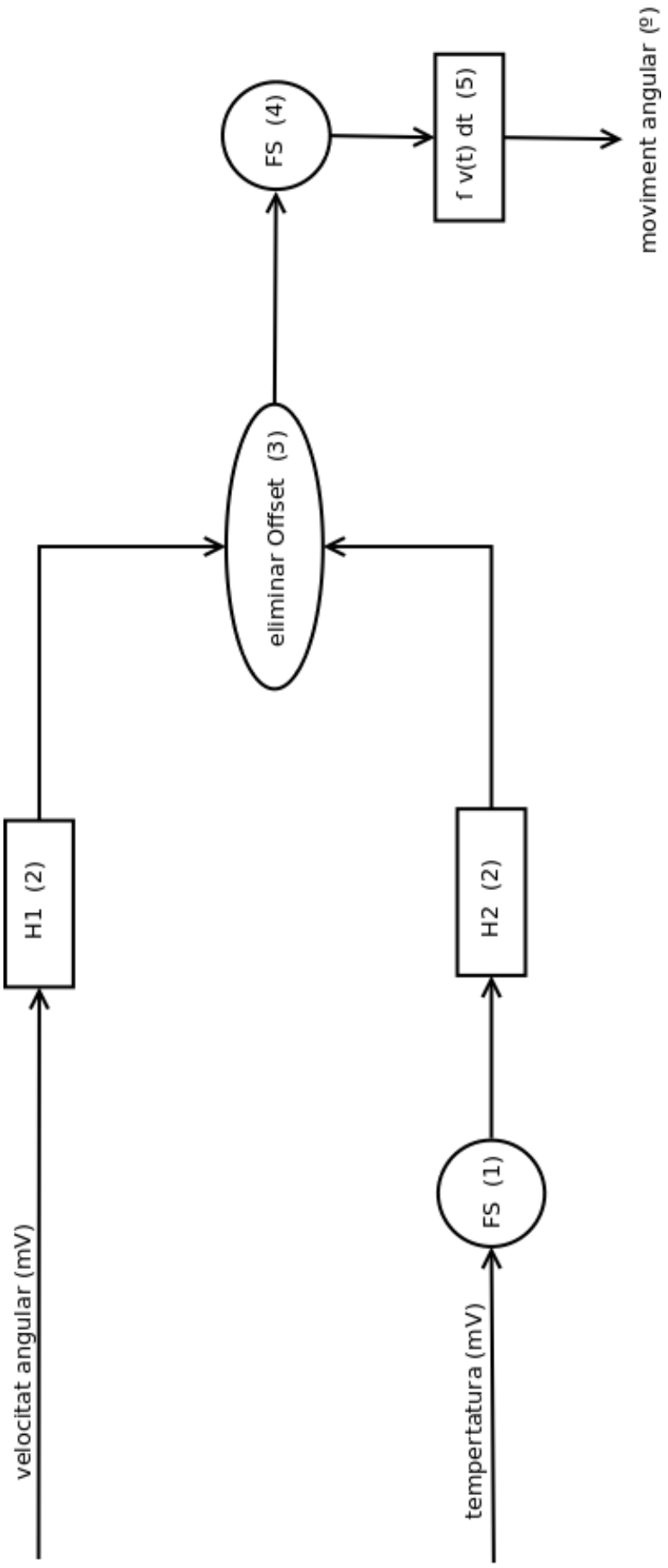


Figura 1.1: Organigrama resum de l'algorisme a implementar

on:

(1) Aquest bloc representa la conversió dels mV revuts de la senyal de temperatura cap a ºC. Segons les especificacions del producte, l'expressió de la recta és $T(V) = 0.009 * T(^oC) + 2.275$, per tant, per obtenir els ºC s'ha d'agafar cada mostra i fer l'operació: $T(^oC) = \frac{T(V) - 2.275}{0.009}$.

(2) En aquest punt, s'han de filtrar les dues senyals mitjançant un filtre pas baix, per tal de reduir soroll i oscil·lacions no desitjades.

(3) Ara ja es pot eliminar l'Offset. Per aconseguir-ho cal restar a la senyal de velocitat (mV) una quantitat d'Offset depenent de la temperatura seguint la següent funció:

$$Offset(T) = p1 * T^9 + p2 * T^8 + p3 * T^7 + p4 * T^6 + p5 * T^5 + p6 * T^4 + p7 * T^3 + p8 * T^2 + p9 * T + p10$$

on:

p1=9.436e-0.19

p2=-2.311e-016

p3=1.335e-014

p4=5.797e-013

p5=-5.635e-011

p6=-5.715e-010

p7=9.954e-008

p8= 5.424e-007

p9=0.0002014

p10=2.5308

i $T^x$ és la temperatura instantània en l'instant $x$.

(4) Conversió de la velocitat angular de volts a º/s. El factor d'escala és 6.16712 mv/º/s, tenint en compte que la freqüència de mostreig és $f_m$ (en Hz), per fer la conversió de velocitat angular en V sense Offset a velocitat angular en º/s cal: $v_{angular}(^o/s) = \frac{v_{angular}(V)}{0.00616712 * f_m}$.

(5) Ara, cada mostra és velocitat angular, per lo que per aconseguir moviment angular, fa falta integrar: $Moviment_{angular}(^o) = \int Velocitat_{angular}(^o/s)dt$.

## 1.3 Solució proposada

La solució que proposem és l'ús del xip MCF5213 de Freescale. Mitjançant l'entorn de desenvolupament (Code Warrior) que ens ofereix aquest fabricant, podem programar, emprant el llenguatge C, l'algorisme que ens permet calcular el moviment angular en cada eix, a partir de la velocitat angular que es captura.

## 1.4 Abast

Un sistema que monitoritzi els moviments d'un objecte per tot l'espai, té moltes aplicacions, la majoria d'elles relacionades directament amb el mercat del transport. A dia d'avui, trobem moltes tecnologies que empren aquests sistemes per dur a terme tasques de control, en podem destacar:

* navegadors GPS
* control de volcat d'automòbils (per exemple, si en un cotxe es detecta un grau d'inclinació superior a lo normal, s'activaran els sistemes de control, com poden ser els airbags)

Tanmateix, la finalitat del sistema en el nostre projecte és diferent a les exposades fins al moment. L'ús pel qual s'ha creat serà el de guiatge d'un sistema autònom, és a dir, que no és governat per cap agent extern a ell. Així doncs, hem de tenir en compte que s'ha d'exigir un nivell de precisió més elevat, ja que aquest no serà controlat per cap persona.

## 1.5 Estructura de la memòria

A grans trets, la memòria està dividida en quatre parts ben diferenciades:

- En el capítol 1 es fa una introducció al projecte a desenvolupar, definint-ne quins són els seus objectius i quin és el punt de partida.
- El capítol 2 es centra en descriure quins són els dispositius físics (el sensor de velocitat angular i el microcontrolador) que ens permeten implementar el sistema descrit en el capítol anterior. No se'n fa una descripció molt exhaustiva, sinó que es detallen aquelles dades més trascendents pel funcionament del nostre sistema. Informacions més específiques hauran de

ser cercades en els manuals que ofereixen els fabricants. A part de la descripció, també hi trobem la justificació de la tria dels dispositius.

- En el capítol 3 es detallen les característiques principals de l'entorn de programació de xips utilitzat: *CodeWarrior*.
- En el capítol 4 s'explica tot el procés de creació de l'aplicació que implementa l'algorisme. En un primer moment es fa una descripció global del sistema que s'ha de desenvolupar. A continuació ja entrem plenament en temàtica d'enginyeria del software, descrivint quin serà el procés utilitzat per tal de programar l'aplicació. Un cop definides les pautes que ens guiaran durant la programació del xip, dividim aquesta en quatre iteracions, en cadascuna de les quals es va afegint alguna millora respecte l'anterior.

Ja per finalitzar, destinem un apartat a citar les conclusions que s'han extret del projecte i la feina que queda a fer en futurs treballs.

# Capítol 2

# Components

## 2.1 Giroscopi

Donat el punt de partida exposat en la introducció, tenim que el giroscopi a utilitzar ja ha estat definit prèviament a l'inici d'aquest projecte.

El giroscopi escollit ha estat el ADXRS610, ja que compleix tots els requisits demanats. La següent taula compara els requeriments mínims amb les característiques del dispositiu escollit.

| **Característica** | **Requisit mínim** | **ADXRS610Z** |
|---|---|---|
| Rang de mesura | ± 200º/s | ± 300 º/s |
| Rang de temperatura | -15ºC a 50ºC | -40ºC a 105ºC |
| Resolució | 0.2º/s | 0.02º/s (depén del soroll) |
| Alimentació | 5 V | 5 V |
| Pes | 50 grams | 0.5 grams |
| Tamany | 10 cc | 0.15 cc |
| Resistència | 10 grams | 2000 grams |
| Preu | 100 euros | 30.56 euros |

Taula 2.1: Comparativa de requisits demanats i especificacions del dispositiu ADXRS610Z

### 2.1.1 Especificacions tècniques

**Sensitivitat**

- Rang de mesura: la màxima velocitat angular que és capaç de captar el dispositiu és ± 300 $^{o}$/s.
- Inicial en funció de la temperatura: el factor d'escala (FS [1]) garantit pel fabricant és del rang 5.52 - 6.48 mv/$^{o}$/s, en funció de la temperatura varia aquest factor de manera no lineal.
- Deriva de la temperatura: el dispositiu té incorporat un sensor de temperatura (per optimitzar el càlcul del FS) que té una deriva de ± "
- No linialitat: l'error del FS utilitzant una línia d'aproximació òptima és del 0.1

**Offset**

- Null: el rang de l'offset [2] garantit pel fabricant és de 2.2 V - 2.8 V.
- Efecte de l'acceleració lineal: l'error produït per una acceleració lineal sobre qualsevol eix és de 0.1$^{o}$/s/g. És un factor a considerar.

**Efecte de soroll**

- Densitat del rang de soroll: indica el marge d"influència que pot tenir el soroll en funció de la freqüència (i l'ample de banda del sistema). El fabricant notifica un soroll típic de $0.05^{o}/s/\sqrt{Hz}$.

**Resposta freqüencial**

- Ample de banda: són les freqüències que poden arribar a composar la senyal de sortida. El dispositiu permet ample de banda de 0.01 Hz a 2500 Hz; aquest paràmetre està controlat per un condensador que actua de filtre pas baix, i en el cas que ens ocupa està limitat a 10 Hz.
- Freqüència de ressonància del sensor: és la freqüència a la qual entre en ressonància el sistema mecànic, entre 12 kHz i 17 kHz.

**Testeig propi (el dispositiu disposa de 2 pins per comprovar el seu correcte funcionament)**

[1] El dispositu retorna una senyal en voltatge, llavors en el cas de tenir un FS= 6 mV/$^{o}$/s, si tenim una senyal constant de +0.872V significa que hi ha una velocitat angular segons l'operació $\Omega = Senyal * \frac{1}{FS}$

[2] Offset o Zero-Rate Output indica la tensió que hi ha a la sortida d'un dispositiu quan teòricament hi ha d'ahaver 0V. En el cas que ens ocupa és la senyal que el ADXRS610Z retorna quan no se li aplica moviment angular. Aquest paràmetre també varia en funció de la temperatura.

- Rang de resposta de ST1 (Self test): és el rang de resposta d'aquest pin on la resposta lògica 0 és de -650 mV a -450 mV i la resposta lògica 1 és de -450 mV a -250 mV.
- Rang de resposta de ST2: és el rang de resposta d'aquest pin on la resposta lògica 0 és de 250 mV a 450 mV i la resposta lògica 1 és de 450 mV a 650 mV.
- Desajustament [3] de ST1 a ST2: segons el fabricant aquest marge d'error té una oscil·lació de ±5%.
- Voltatge d'entrada d'1 lògic: és el voltatge que s'ha d'aplicar a un pin per rebre una resposta d'un 1 lògic. Aquest rang és de 3.3V a 5V.

**Sensor de temperatura**

- $V_{OUT}$ (sortida de temperatura) a 25 °C: de 2.35 V a 2.65 V és el rang de voltatge que s'obté del dispositiu a 25 °C, tot i que el voltatge típic és de 2.5V. S'ha de considerar una impedància de sortida $\geq$ 10MΩ.
- Factor d'escala: a 25 °C i $V_{Ratio}$ = 5 V (és la alimentació del sensor de temperatura) és de 9 mV/°C.
- Impedància d'entrada a $V_S$ (entrada d'alimentació) i entrades comunes ($AV_{CC} = V_{DD} = V_S = V_{Ratio}$ = 5V): típicament de 25 kΩ.

**Retard d'arrencada**

- És el temps que tarda el dispositiu en ésser operatiu després d'alimentar-lo, està situat sobre els 50 ms com a màxim.

**Propietats de la sortida de senyal**

- Intensitat màxima: 200 $\mu$A.
- Capacitància de càrrega màxima: 1000 $\rho$F.

**Alimentació**

- Voltatge operatiu: rang de 4.75 V a 5.25 V, tot i que el típic és de 5 V.

[3] El desajustament del testeig propi es defineix com $2 * \frac{ST2+ST1}{ST2-ST1}$

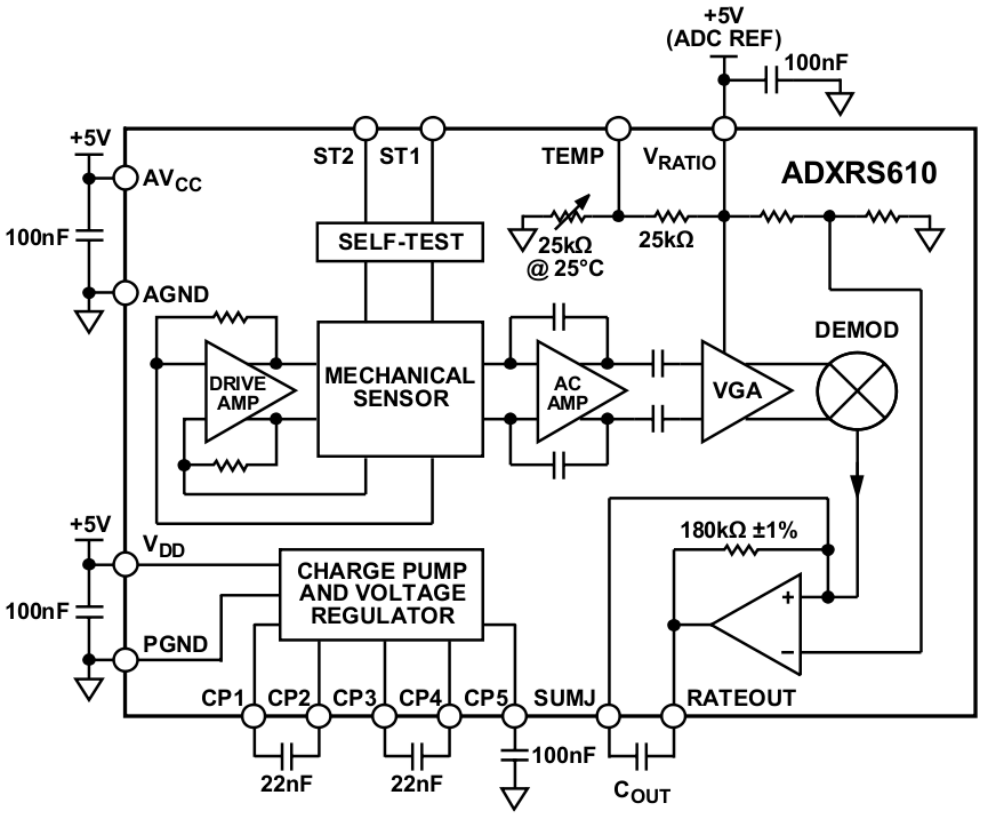


Figura 2.1: Diagrama de blocs funcionals del dispositiu ADXRS610

- Intensitat en $0^o/s$: la intensitat que consumeix el dispositiu sense moviment és típicament de 3.5 mA, arribant a un màxim de 4.5 mA.

### 2.1.2 Visió global del dispositiu

El ADXRS610Z és un dispositiu que integra el ADXRS610 amb la circuiteria necessària pel seu bon funcionament. El diagrama de blocs funcionals del ADXRS610 és el de la figura 2.1.

L'esquema del ADXRS610Z és el mostrat en la figura 2.2, on hi ha especificats els pins, la funció dels quals queda detallada a la taula 2.2.

Per tal d'evitar que la senyal que s'obté del sensor no sigui influenciada per factors externs, el ADXRS610 té implementada tota l'electrònica al mateix xip que el sensor. Això fa que el dispositiu implementat sigui d'un tamany molt reduït i d'una gran eficiència. Els elements electrònics necessaris són els següents:

- Bomba de càrrega: permet aconseguir una tensió màxima de 18 Volts

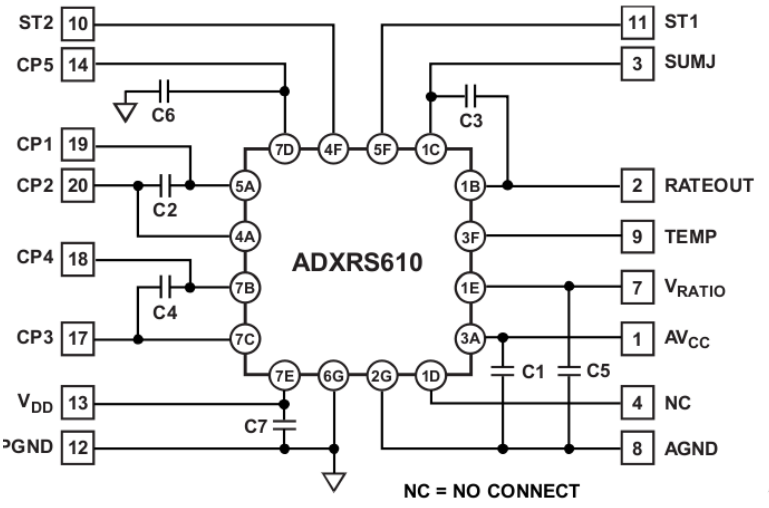


Figura 2.2: Esquema del dispositiu ADXRS610Z

- Drive AMP: és un amplificador disposat de manera que manté els dos sensors mecànics desacoplats.
- DEMOD: és un circuit demodulador necessari per aconseguir que la senyal que surt del sensor mecànic (que està modulada) es converteixi en tensió analògica contínua.
- TEMP: és un circuit que permet captar la temperatura del dispositiu per aconseguir compensar la diferència de la temperatura en el moviment angular final.

A part d'aquests dispositius, hi ha un seguit d'amplificadors de senyal per tal de poder processar la senyal de velocitat.

**Observació**: s'ha obviat quin és el funcionament mecànic del dispositiu, donat que no aporta rellevància al desenvolupament del projecte. Tot i això, en podem obtenir la informació necessària en el ja citat, en l'apartat 1.2, treball de final de carrera d'en Robert Rocamora, d'on també s'han extret les dades exposades en aquest apartat del sensor ADXRS610.

| Nº de PIN | Etiqueta | Descripció |
|---|---|---|
| 1 | $AV_{CC}$ | Alimentació positiva (+5V) |
| 2 | $RATEOUT$ | Senyal de sortida analògica de velocitat |
| 3 | $SUMJ$ | Senyal d'amplificació i tall d'ample de banda de $RATEOUT$ |
| 4 | $NC$ | No connectada |
| 5 | $NC$ | No connectada |
| 6 | $NC$ | No connectada |
| 7 | $V_{RATIO}$ | Alimentació de referència per $RATEOUT$ (+5v) |
| 8 | $AGND$ | Massa de l'alimentació |
| 9 | $TEMP$ | Senyal de sortida analògica de temperatura |
| 10 | $ST2$ | Testeig propi del sensor 2 |
| 11 | $ST1$ | Testeig propi del sensor 1 |
| 12 | $PGND$ | Massa de la bomba de càrrega |
| 13 | $V_{DD}$ | Alimentació de la bomba de càrrega (+5V) |
| 14 | $CP5$ | Condensador de 100 nF (limita l'ample de banda a 10 Hz) |
| 15 | $NC$ | No connectada |
| 16 | $NC$ | No connectada |
| 17 | $CP3$ | Condensador de 22 nF (per la bomba de càrrega) |
| 18 | $CP4$ | Condensador de 22 nF (per la bomba de càrrega) |
| 19 | $CP1$ | Condensador de 22 nF (per la bomba de càrrega) |
| 20 | $CP2$ | Condensador de 22 nF (per la bomba de càrrega) |

Taula 2.2: Descripció de la funcionalitat dels pins del ADXRS610Z

## 2.2 Xip

A l'hora d'escollir el xip o microcontrolador més adequat a les nostres necessitats, hem de tenir en compte una sèrie de camps: nº de bits, tipus de memòria, quantitat de memòria, velocitat a la que treballa, mòduls de què disposa, ...

Sabent quina és l'envergadura del problema que tenim entre mans, podem començar a definir els valors d'alguns dels paràmetres anteriors, per tal de fer una cerca més acurada dels xips que hi ha al mercat que millor s'ajusten als nostres requeriments. Així doncs, podem fixar:

Nº bits: 32 bits

Tipus de memòria: FLASH i RAM

Quantitat mínima de memòria FLASH: 128 KB

Observació: per tal de dur a terme la implementació de la solució proposada, no necessitem un microcontrolador de 32 bits, sinó que amb un PIC ja ho podríem realitzar. De totes formes, s'ha decidit emprar un xip que treballi amb registres de 32 bits, perquè en un futur, aquest haurà de realitzar moltes més tasques que no pas les especificades en aquest projecte, donat que forma part d'un treball de major envergadura que cal tenir en compte. En aquell moment, s'hauria de deixar de treballar amb un PIC per passar a fer-ho amb un microcontrolador, provocant així, que els programes que s'implementen en aquest projecte perdessin el seu valor.

Amb aquesta informació, podem utilitzar les diferents eines que ofereix cada fabricant per conèixer els xips que disposa que compleixin aquestes característiques. Un cop realitzada aquesta cerca, els dos xips que s'adapten millor a les nostres necessitats són:

- Texas Instrument. LM3S9B95 Microcontroller
- Freescale. Família MCF521X

Cal dir, que de xips que satisfan els requeriments n'hi ha molts. S'han descartat tots aquells que a part d'aquestes característiques, n'afegien de noves i milloraven les existents amb escreix, donat que això suposava un augment del preu i del tamany del microcontrolador, per aspectes d'aquest que no haguéssim utilitzat.

### 2.2.1 Xip LM3S9B95

Les característiques principals del xip considerades útils per tal d'implementar la solució al nostre problema són:

32 bits

256 KB memòria flash

96 KB memòria SRAM

Freqüència (MHz): 80 i 100

3 UARTs

2 I2C

2 SSI (synchronous serial interface)

1 I2S (so)

Mòdul controlador DMA

4 timers de 32 bits

2 conversors analògic-digital de 10 bits

Interrupcions vectoritzades

3 comparadors analògics

16 comparadors digitals

-40 a 85º

Pulse width modulator

100 patilles

### 2.2.2 Família MCF521X

Donat que són molt similars tots els xips de la família i que el que tenim més a l'abast és el MCF5213, serà aquest el microcontrolador del qual se'n detallaran les seves característiques bàsiques:

32 bits

256 KB memòria FLASH

32 KB SRAM

Freqüència: 66 i 80 MHz

3 UARTs

1 I2C

Mòdul controlador DMA

4 timers 32 bits de propòsit general

2 timers programables d'interrupcions

1 conversor analògic-digital de 12 bits

1 mòdul d'E/S de propòsit general

Pulse width modulator

100 patilles

Tal i com queda palès en les descripcions dels dos xips que comparem, podem veure que, en termes generals, el de Texas Instrument és un xip més complert que el de Freescale. Tanmateix, donat que el MCF5213 compleix amb totes les nostres necessitats:

- FLASH suficientment gran per emmagatzemar-hi el programa
- Mòdul UART
- Mòdul I2C
- Conversor analògic-digital
- Timers de propòsit general (GPT)

i que el conversor analògic-digital, que és el mòdul principal utilitzat en la solució proposada, treballa amb registres de 12 bits, enlloc de 10 bits com ho fa el de Texas Instrument, finalment hem decidit escollir-lo com a xip a utilitzar per tal d'implementar el sistema.

Un altre aspecte a considerar és que aquest treball s'ubica dins d'un projecte de major envergadura, en el que ja s'han utilitzat microcontroladors MCF5213. S'ha de valorar que en el moment que hagin de treballar conjuntament, dins del sistema global, la seva interconnexió sempre serà més senzilla que si utilitzem xips de fabricants diferents. Així doncs, es tracta d'un xip més accessible i del que ja en tenim referències.

També s'ha valorat molt positivament que l'entorn de programació que ofereix Freescale per la programació dels seus xips (*CodeWarrior*) ja ens era conegut, permetent-nos programar el microcontrolador amb llenguatges de programació també coneguts, com són el C i C++, en cas de necessitar utilitzar programació orientada a objectes.

**Freescale ColdFire MCF5213**

Com ja s'ha comentat en l'apartat anterior, definitivament el xip escollit és el MCF5213 de la casa Freescale. No en donarem més característiques, ja que aquestes ja ens són irrellevants en el nostre projecte. El que sí que és necessari i ens pot ajudar a entendre'n el seu funcionament és el seu diagrama de blocs.

En l'annex A, hi trobem detallada la funcionalitat de cada PIN.

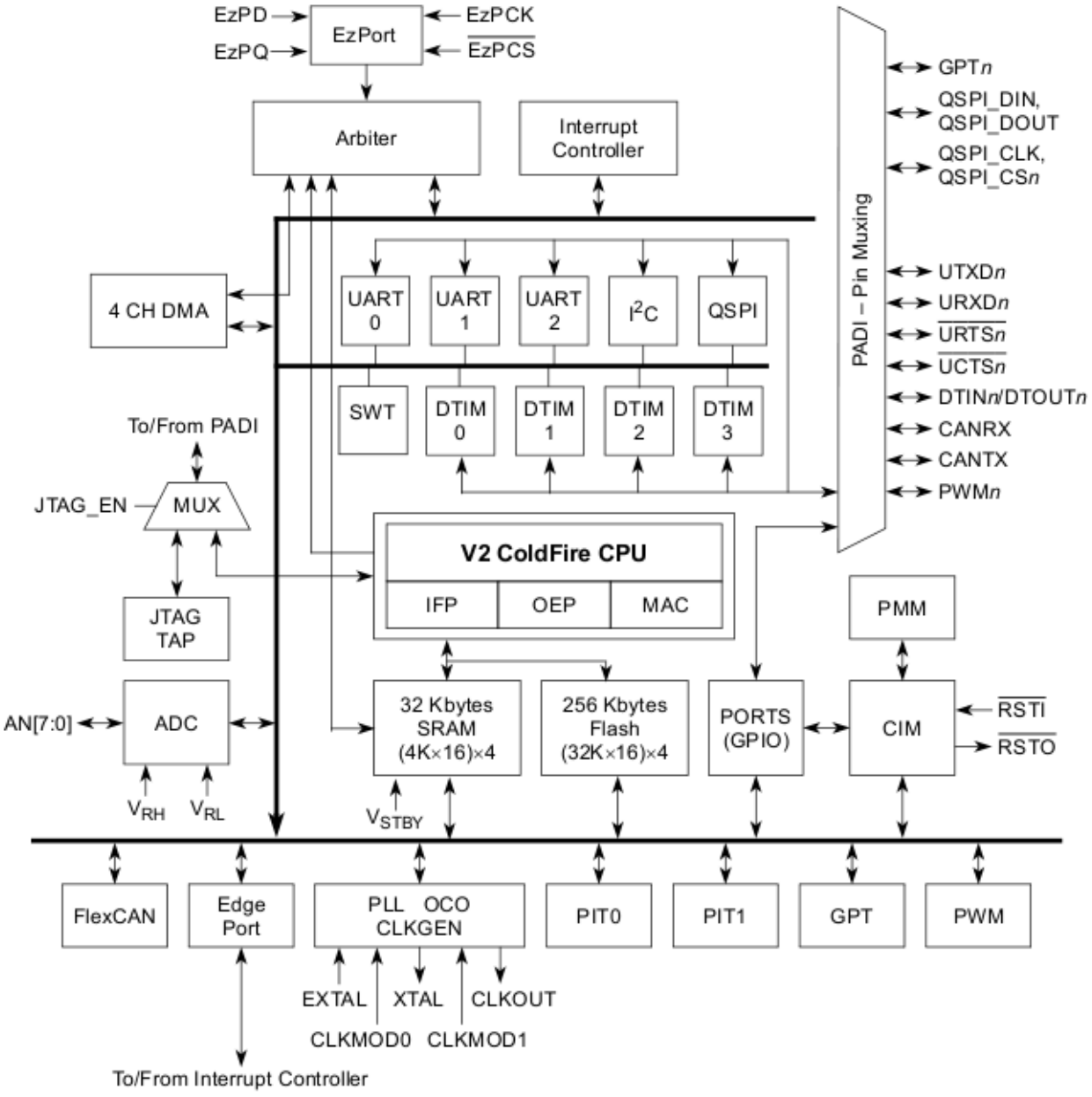


Figura 2.3: Esquema del dispositiu MCF5213

# Capítol 3

# Entorn de programació de xips: CodeWarrior

> Observació: informació extreta del treball final de carrera de n'Albert Agraz Sánchez i dirigit per en Francisco Clariá Sancho, titulat "Processat de senyals d'acceleració i metodologia d'ajust de zero".

El fabricant Freescale (com a part de Motorola) ens ofereix un entorn de programació professional dedicat als xips que fabrica. Disposa d'una versió general que s'adapta a les diferents famílies de processadors amb els que tracta. Aquest entorn s'anomena CodeWarrior.

Concretament nosaltres disposarem d'una versió de CodeWarrior específicament adaptada per a la família de processadors ColdFire. Aquesta edició disposa de tot el necessari per a crear una aplicació enfocada a aquesta arquitectura.

Sovint, els programadors que no estan acostumats a treballar amb sistemes embedits i no paren atenció, en què una de les grans virtuts dels entorns de programació integrats és el fet que et permeten escriure en el xip amb només un parell de clics. És el cas del CW que ens ofereix una eina per esborrar i escriure el nostre MCF5213 sense cap complicació. A més, disposem d'una funció que integra la compilació, l'enllaçament i l'escriptura en el xip.

Aquesta independència entre el codi i l'arquitectura a la que va dirigit, ens aporta un benefici en quant a consistència. La consistència en aquest cas s'entén com la separació que tenim entre implementació i hardware sobre el que s'ha implementat.


Figura 3.1: Versió del programari CodeWarrior emprat

## 3.1 Cicle de programació

El propi fabricant ens indica quin és el cicle de programació que es segueix em el seu entorn. Aquest és exclusivament orientat a l'ús del seu entorn, per tant, no considera en aquesta descripció fases tant importants com el disseny de l'arquitectura o l'anàlisi de requeriments:

- Tenir una idea per a un nou software
- Implementar la idea en codi
- Compilar el codi
- Enllaçar el codi
- Corregir els errors
- Produir una versió final del producte

No podem confondre aquest cicle de programació amb el cicle de desenvolupament de software (que s'explicarà en el capítol següent). En aquest apartat és fa referència únicament a aquelles parts del procés de desenvolupament de software que estan relacionats amb l'eina CodeWarrior.

Les principals funcionalitats que ens ofereix el CodeWarrior són les següents:

- **Crear (create)**: es tracta d'incialitzar el projecte, amb la configuració específica del xip i el codi software que ens permet aplicar la configuració general al MCF5213.

- **Editar (edit)**: afegir el codi propi a l'estructura del projecte creada per CW. Aquí és on farem que l'aplicació faci el que nosaltres volem. Podrem crear funcions, interfícies, classes, rutines, i tot el que hem vist en llenguatges d'alt nivell.
- **Compilar (compile)**: es tracta de convertir el codi d'alt nivell en un llenguatge màquina que s'executi en el nostre xip. Transformarà instrucció a instrucció, així com afegira les interrupcions necessàries a la taula, etc.
- **Enllaçar (link)**: generarem un únic fitxer binari executable per el host destí. Aquest serà el fitxer que copiarem a la memòria Flash del dispositiu.
- **Depurar (debug)**: és la part en la que podrem observar pas a pas l'execució del nostre codi, així com els valors de les variables, les crides a funcions, etc. Fent ús d'aquesta eina podrem saber on fallen les nostres aplicacions i corregir aquests errors.
- **Produir versió final (release)**: quan haguem acabat el desenvolupament de l'aplicació, farem una versió final del producte ja empaquetada i llesta per a copiar al dispositiu i executar.

## 3.2 Components del Code Warrior

L'entorn de desenvolupament CodeWarrior ens ofereix diverses eines per a realitzar tot allò que vulguem amb el nostre producte software. Són eines que ens ajuden des de l'organització fins a la realització de proves.

En concret, les eines que ens ofereix l'entorn (ja sigui a través d'accessos o bé directament sobre el IDE) són les següents:

- Gestor de Projectes
- Editor
- Motor de Cerca
- Navegador de Codi
- Sistema de Construcció
- Depurador
- Programador del Xip

Les cinc primeres eines són comunes en qualsevol entorn de programació integrat, i el seu funcionament és molt similar entre aquestes. Així doncs, veurem amb més detall les dues darreres.

### 3.2.1 Depurador

El depurador (*debugger* en anglés) és l'eina que ens permet detectar errors en el nostre codi. La clau principal del seu funcionament és l'execució individual de sentències de codi, és a dir, poder executar les instruccions una a una, sense haver d'esperar al final de l'execució per a veure el valor de les variables.

Per al control més avançat de l'execució podem definir punts de parada (o *breakpoints*), és a dir, punts en que el depurador pararà l'execució del programa per a que puguem veure l'estat actual del procés. Indicarem aquets punts de parada en el propi codi, marcant la fila en concret amb un punt vermell.

Gràcies al seu analitzador de variables, podem veure en cada instant el valor de totes les variables actives en l'execució d'una rutina. A més a més, podem escollir el format en el que volem veure els valors, des de enters, fins a hexadecimal, passant per binari o caràcter.

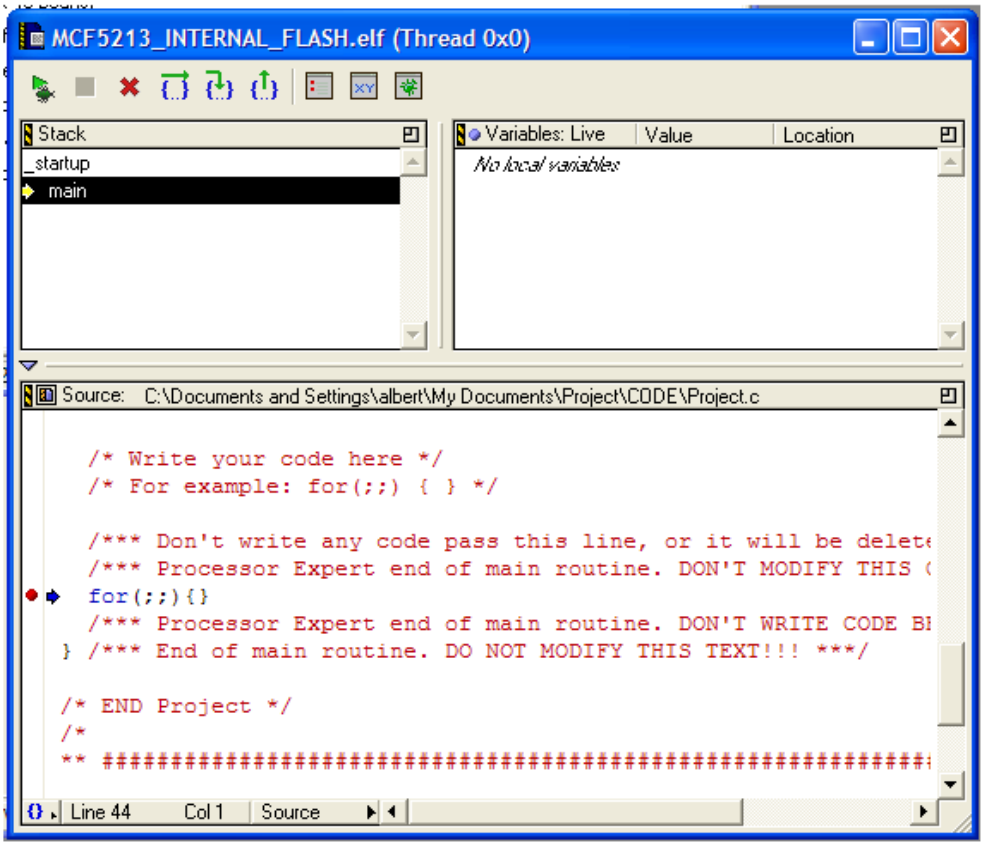


Figura 3.2: Depurador de l'entorn CodeWarrior

### 3.2.2 Programador de xips

Un cop tenim una aplicació construïda (compilada i enllaçada) és el moment de passar-la al processador per tal de provar-la o bé depurar-la.

Amb una senzilla finestra (figura 3.3), ens permet indicar quin processador és el que estem programant i les dades sobre la memòria (adreça d'inici, adreça de fi, etc).

A través de les pestanyes del costat accedirem a la totalitat de funcionalitats que ens ofereix:

- Esborrar la memòria.
- Comprovar que la memòria està en blanc.
- Escriure un programa.
- Comprovar que un programa ha estat escrit.
- Realitzar una comprovació d'integritat.

Un cop seleccionat què volem fer, l'eina s'encarregarà de la resta (establir comunicació USB, codi de connexió, etc.).

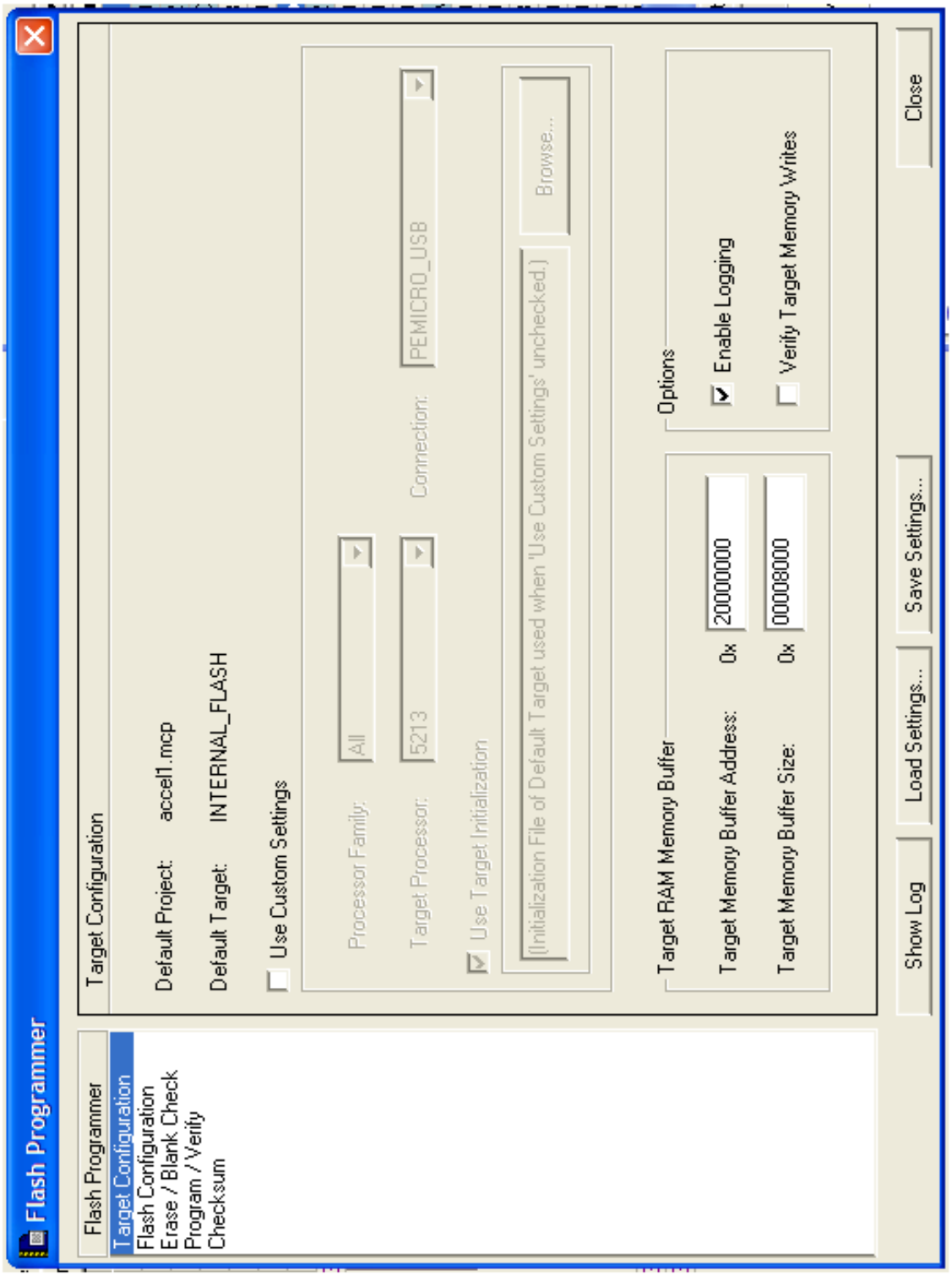


Figura 3.3: Menú de programació de xip en l'entorn CodeWarrior

## 3.3 Processor Expert

Desenvolupar aplicacions per a microcontroladors implica invertir molt temps en configurar els diferents mòduls utilitzats. Aquesta configuració es fa a través de la manipulació dels registres del processador, feina que resulta molt feixuga i pot acabar en moltes ocasions en un mal funcionament de l'equip.

Però no només per a configurar els mòduls es fa necessari treballar amb registres. Per qualsevol interacció amb el mòdul (ja sigui de configuració o tractament de dades) implica treballar a baix nivell. Per tant, si volem llegir dades d'un mòdul o bé enviar-ne, haurem d'emprar registres.

Per tal d'evitar aquest fet, utilitzarem l'eina **Processor Expert** que ens ofereix el *CodeWarrior* que:

- Ens permet inicialitzar i configurar els mòduls amb els paràmetres desitjats (introduïts mitjançant un entorn gràfic).
- Ofereix una llibreria per cada mòdul
- Incorpora un gestor d'interrupcions més senzill.

Així doncs, partir d'ara ja no parlarem de mòduls, sinó de **beans**. Un *bean* és la configuració, les funcions de la seva llibreria associada i els events relacionats amb el mòdul.

El Processor Expert disposa del seu propi entorn de funcionament on apareixen els beans afegits al projecte amb les seves funcions disponibles i els possibles events. Les funcions ens permeten interactuar amb les dades del nostre mòdul. Els events són les interrupcions que genera el mòdul, però expressades en forma de funció. En un principi, aquesta estarà buida, i hi podrem afegir el codi necessari per tal de tractar el succés.

Un dels avantatges de treballar amb el Processor Expert és que podem activar o desactivar tant funcions com events. El fet d'activar o desactivar aquests ítems ens reduirà el codi generat per l'eina, i per tant ens permetrà encabir més codi d'usuari en el xip.

# Capítol 4

# Desenvolupament

## 4.1 Visió global del sistema

Abans d'endinsar-nos en el desenvolupament software del sistema, és important donar una idea global d'aquest, especificant-ne que és el que realment s'està implementant.

L'objectiu del projecte és capturar, a través del sensor ADXRS610, les velocitats angulars i temperatures de cadascun dels tres plans ortogonals que conformen l'espai i calcular-ne (mitjançant el microcontrolador) el moviment angular produït per un mòbil en cada eix, per tal de conèixer el punt exacte de l'espai on es troba ubicat.

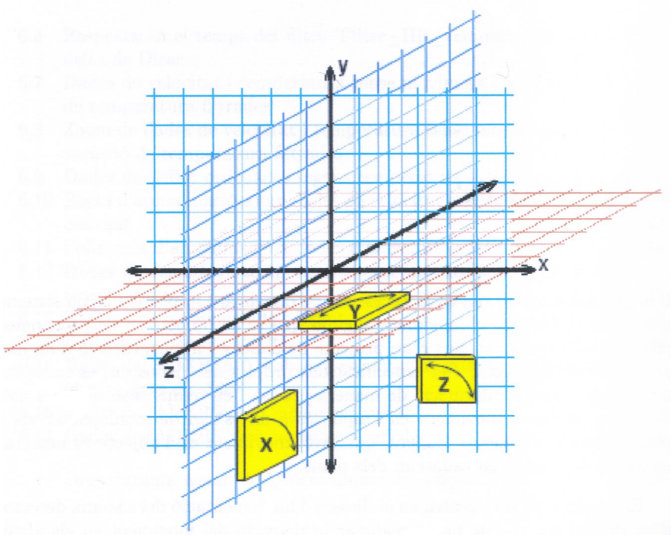


Figura 4.1: Representació dels 3 plans on es pot enregistrar moviment angular

Així doncs, podem dividir el sistema en dos fases ben diferenciades:

1. Captura dels valors obtinguts amb el sensor ADXRS610, en els tres plans
   - Velocitat angular
   - Temperatura
2. Càlcul del moviment angular

Captura dels valors obtinguts amb el sensor

El que ens ajudarà a fer més comprensible el funcionament d'aquesta fase n'és el seu diagrama de blocs (figura 4.2).

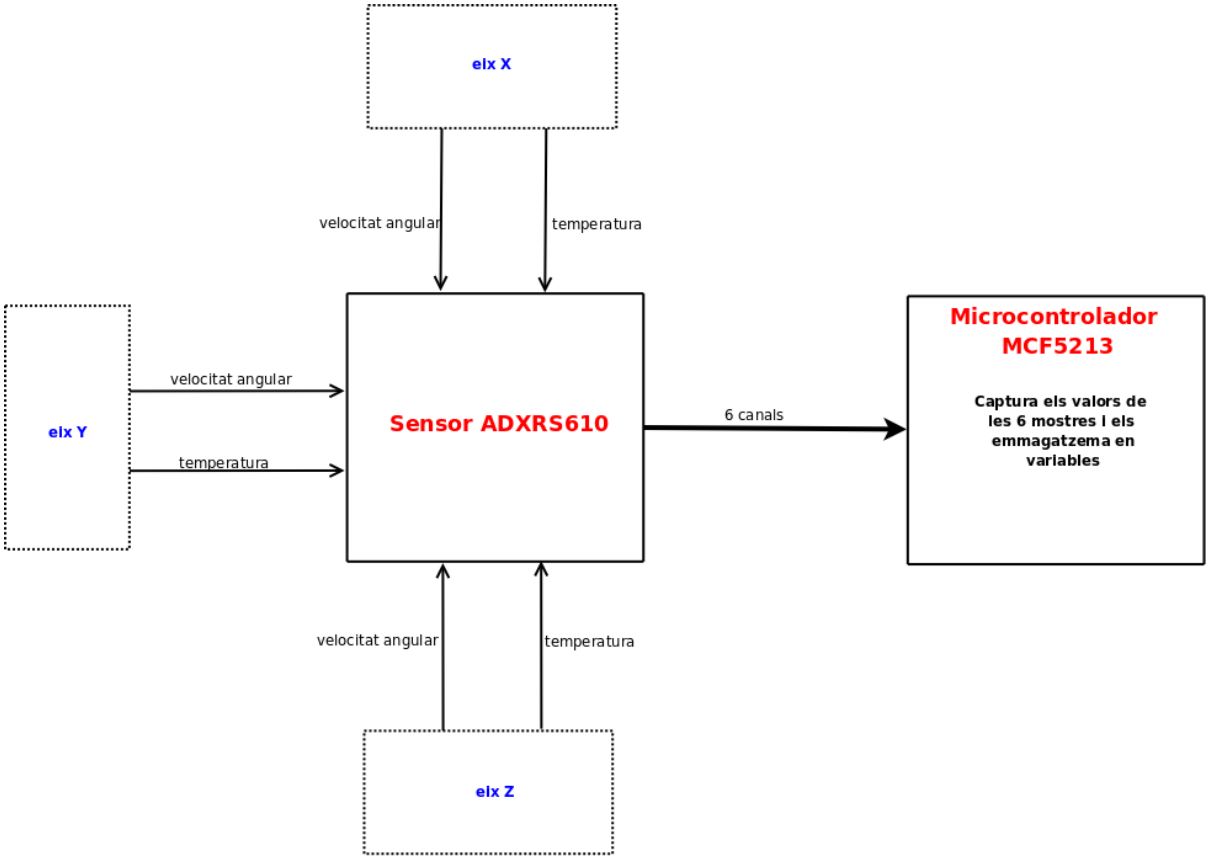


Figura 4.2: Diagrama de blocs de la captura de dades obtingudes amb el sensor

Càlcul del moviment angular en cada eix

En aquest segon fragment es programa el microcontrolador per tal de calcular, a partir de les dades obtingudes en la fase anterior, el moviment angular en cadascun dels eixos. En aquest cas, tots els

càlculs es produeixen en el xip, sense interaccionar amb cap altre dispositiu. Tanmateix, el següent diagrama ens pot ajudar a entendre millor que és el que estem desenvolupant en aquesta etapa.

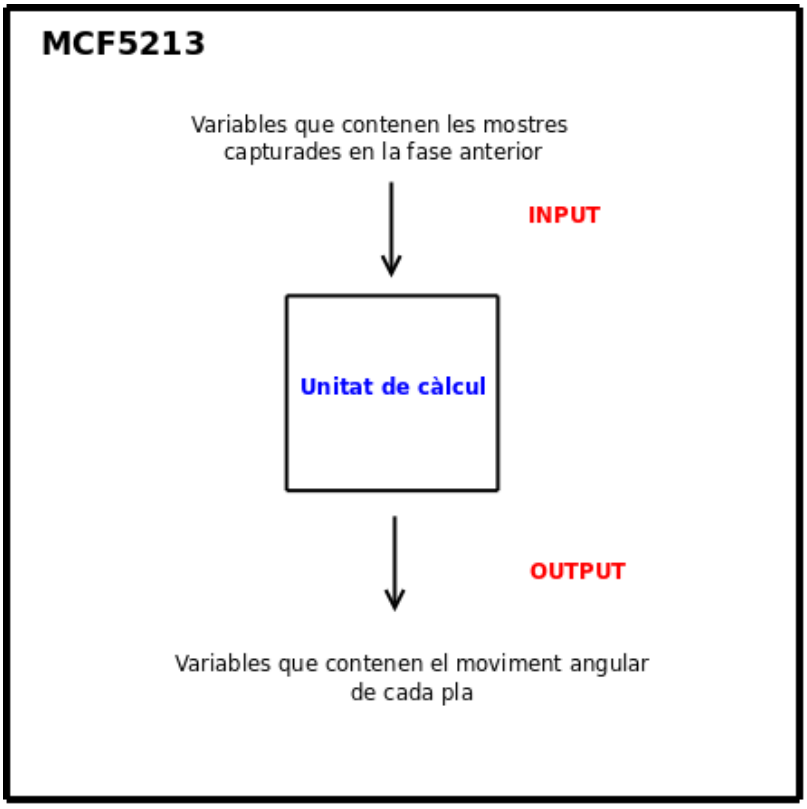


Figura 4.3: Diagrama del càlcul del moviment angular

Cada cop que es reben les 6 mostres, es duu a terme aquest procés, és a dir, les variables que contenen els valors de les mostres s'actualitzen (no es creen noves variables) al nou valor capturat, es calcula el moviment angular de cada pla, i s'actualitza el valor de les variables que conté aquesta informació.

Tal i com es pot apreciar en la figura, els resultats calculats (moviment angular de cada pla) resten emmagatzemats en variables.

## 4.2 Model de desenvolupament

Un model de procés de software és una descripció bàsica i genèrica d'un procés de desenvolupament de software, que en proporciona una representació abstracta.

Un procés de desenvolupament és un conjunt d'activitats i resultats associats que condueixen al desenvolupament d'un producte software a partir dels requisits d'un usuari. Les activitats principals d'un model són:

- Anàlisi
- Desenvolupament
- Validació
- Manteniment

Tot i que l'aplicació a desenvolupar no sigui d'una gran envergadura, sempre és important recolzar-se amb un model, és a dir, disposar d'unes pautes que estructurin el procés de creació d'una aplicació.

Dels molts models que existeixen actualment en enginyeria del software, el més adequat és emprar un model evolutiu. Un model evolutiu divideix la creació d'un programa en diferents fases (o iteracions), cadascuna de les quals està formada per les etapes d'anàlisi, desenvolupament, validació i manteniment.

Es basa en la creació d'una primera versió de l'aplicació, i a partir d'aquesta es va refinant, aplicant els canvis que millorin el programa, o en solucionin els possibles errors que hi poguessin haver en iteracions anteriors.

Un avantatge que ens ofereix el fet d'utilitzar un model evolutiu és que podem dividir la funcionalitat final del sistema, en fragments de funcionalitats més petits, i treballar cadascun d'aquests en fases diferents, facilitant-ne la programació, la correcció d'errors i la depuració de l'aplicació.

També cal valorar molt positivament, que utilitzar aquest model ens permet que els requeriments dels sistema siguin força dinàmics, ja que començarem a desenvolupar l'aplicació pels fragments que estiguin més ben especificats, i prorrogarem per a properes iteracions aquelles funcionalitats que no estiguin tan ben definides, o sigui, que no en coneixem els requeriments amb exactitud, o que probablement aquests hagin de variar en funció de les decisions que es prenguin en cada fase.

De models evolutius n'hi ha m'és d'un. En el nostre cas, el que millor s'ajusta al sistema que tenim entre mans, és l'incremental. L'avantatge principal que ens aporta és l'entrellaçament d'iteracions, és a dir, no cal esperar a completar totalment una fase, per tal d'iniciar la següent. Això no vol dir que puguem començar totes les iteracions alhora, sinó, que per exemple, un cop realitzada l'etapa

de validació d'una iteració, podem iniciar l'anàlisi de la següent fase, i anar-les desenvolupant concurrentment.

## 4.3 Iteracions

Tal i com s'ha explicat en l'apartat de model del procés de software, dividir el desenvolupament del sistema en diverses iteracions ens en facilitarà la tasca. En un primer moment, ens permet desglossar el problema global en fragments més petits, i per tant més senzills d'implementar i depurar.

L'objectiu de cada nova iteració és millorar algun dels aspectes de les anteriors. El terme millorar és molt ampli, i pot incloure moltes accions, com ara: afegir alguna nova funcionalitat, solucionar errors que s'han produït en iteracions anteriors, augmentar-ne el rendiment, disminuir l'espai de memòria a utilitzar, tenir en compte casos molt concrets que s'havien menystingut en fases anteriors,...

Així doncs, s'ha dividit el desenvolupament de l'aplicació en quatre iteracions.

- Primera iteració. En un primer moment, només ens proposem capturar dades des d'un generador de funcions, operar amb aquestes i mostrar els resultats, per tal de comprovar que la lectura s'ha fet correctament.

- Segona iteració. Un cop s'hagi ratificat que la lectura de dades és correcta, podem capturar i operar amb les dades d'un pla. El motiu de fer-ho en un únic pla, és simplificar el problema i evitar la propagació d'errors per triplicat. Una vegada s'hagi assolit aquest pas, ja es podrà extrapolar als altres dos plans restants.

- Tercera iteració. L'objectiu de la tercera iteració és capturar les dades dels sis canals de lectura i poder operar amb elles sense tenir en compte en cap moment el rendiment de l'aplicació ni la memòria que aquesta necessitarà per tal d'executar-se.

- Quarta iteració. En aquesta darrera fase es tracta de refinar el programa, sense afegir cap funcionalitat nova. L'objectiu és alliberar el màxim de memòria possible, simplificant la quantitat de variables que s'usen, reutilitzant-ne algunes i el·liminant-ne d'altres d'innecessàries que només utilitzàvem en les iteracions anteriors per tal que el programa fos més comprensible i facilitar-ne la detecció d'errades i depuració.

### 4.3.1 Primera iteració

#### Anàlisi

La finalitat d'aquesta primera iteració és realitzar la lectura de dades pels pins que en futures iteracions ens permetran capturar informació dels sensors. Aquests són: AN0, AN1, AN2, AN3, AN4 i AN5. Per no complicar més aquesta fase, fer més senzilla la detecció dels possibles errors que es puguin produir i poder comprovar la correctesa dels resultats obtinguts, la senyal d'entrada provindrà d'un generador de funcions.

#### Desenvolupament

El desenvolupament d'aquesta fase és força senzill. S'ha de connectar la senyal que ens proporciona el generador de funcions amb el pin a través del qual desitgem fer la lectura. Cal tenir en compte les amplituds d'aquestes senyals, ja que si aquestes fossin molt elevades podrien malmetre el xip. Per evitar-ho, no treballarem amb tensions superiors als 5 volts. També cal tenir en compte que s'han de connectar totes les masses al pin GND.

Per realitzar la lectura utilitzem una instància del bean ADC (conversor analògic-digital) amb nom AD1. En el fitxer base (main) del programa, n'iniciem la seva execució, emprant la instrucció:

```
AD1_Start();
```

Un cop ja està en funcionament hem de realitzar la lectura d'una mostra de la senyal a cada event OnEnd del AD1. Per fer-ho, cal editar el fitxer que controla els events (events.c):

```
...
word* value;
double captura;
...
void AD1_OnEnd(void){
    AD1_GetChanValue16(canal, value);
    captura=(double)((*value>>4)*5/4096);
}
```

on canal és un nombre enter entre 0 i 5 que especifica per quin dels 6 canals que haurem d'utilitzar

per capturar dades dels tres plans, en iteracions posteriors, volem obtenir la mostra.

A l'hora de capturar la mostra pel pin escollit, internament, s'utilitzen registres de 12 bits. Per tal d'obtenir el valor real, haurem de:

- Desplaçar quatre bits el resultat obtingut.
- Multiplicar per 5 (valor màxim) i dividir entre 4096 ($2^{12}$)

Fins al moment, *value* era una variable entera, però al fer aquesta divisió, passa a ser un nombre real, per tant, cal especificar aquesta conversió amb el *cast* (*double*).

En aquest punt, ja tenim el valor de la mostra emmagatzemada en la variable *captura*. Aquesta s'ha creat fora de l'event, perquè no calgui crear una variable cada cop que és llegeixi una mostra, sinó que només se n'haurà de modificar el seu valor.

Un cop ja tenim capturat el valor, el que ens interessa ara és poder enviar-lo pel port sèrie i rebre'l a la Hyperterminal del nostre ordinador. Per fer-ho necessitem crear una instància del bean AsynchroSerial, amb la següent configuració:

Parity: none

Width: 8 bits

Stop bit: 1000

Baud rate: 19200 baud

En un primer moment, per realitzar aquesta transferència utilitzàvem la funció *sendBlock* del bean esmentat. Aquest fet afegia noves dificultats i complicacions al programa; per tant, es va decidir que en les primeres iteracions, en què no tenim en compte el rendiment de l'aplicació, no faríem ús d'aquesta funció i empraríem les llibreries del llenguatge de programació C i les pròpies de l'eina de desenvolupament *Code Warrior*. Això va comportar que s'havien de modificar les llibreries, eliminant-ne les funcions que no ens eren necessàries, per tal de no sobrepassar la memòria FLASH del xip.

Així doncs, per tal d'enviar les dades pel port sèrie s'utilitza la instrucció *printf*.

```
printf(\%i\n",*value);
```

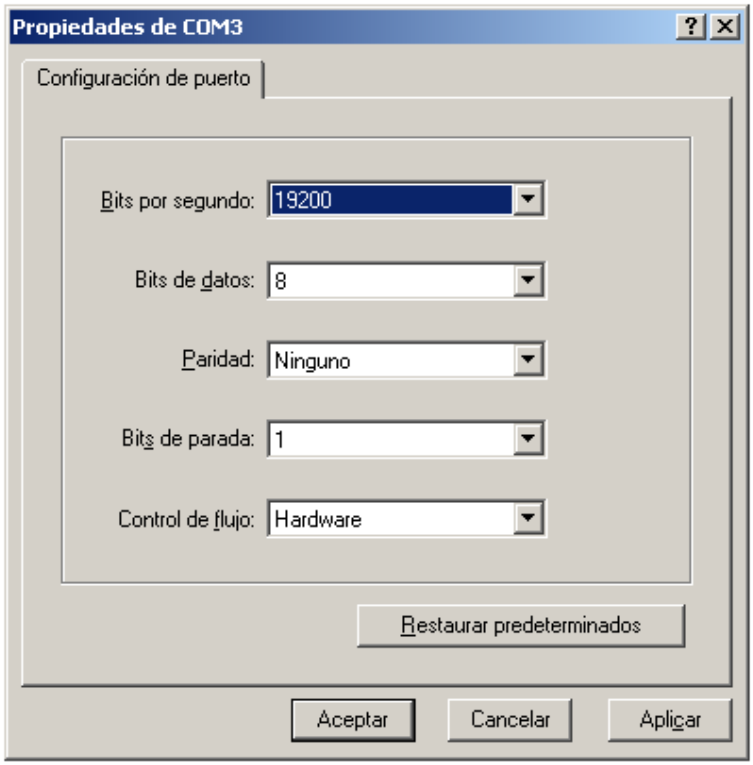


Figura 4.4: Configuració HyperTerminal

Finalment, cal configurar els paràmetres del programa Hyperterminal (figura 4.4), aplicació mitjançant la qual rebrem les dades al ordenador, especificant els mateixos valors que els utilitzats en el bean *AsynchronSerial*.

### Validació

Per tal de comprovar la correctesa dels resultats obtinguts, el que fem és utilitzar una aplicació que ens permeti dibuixar gràfics a partir de les dades capturades i el comparem, utilitzant un oscil·loscopi, amb el senyal que ens ha proporcionat el generador de funcions.

Els resultats obtinguts els considerem bons. Això ens permet iniciar una nova iteració.

### Manteniment

Aquesta fase no contempla una etapa de manteniment. Els resultats obtinguts s'han considerat com a correctes. Això no vol dir que la captura de dades no pugui ser millorada, sinó que en cas de ser-ho, ja es desenvoluparia en una iteració posterior.

### 4.3.2 Segona iteració

#### Anàlisi

Un cop ja capturem les mostres de la senyal, el següent pas és començar a implementar l'algorisme que ens permet calcular el moviment angular d'un pla, especificat en l'apartat Punt de Partida.

Només és desenvoluparà el sistema per un pla, i un cop hagi estat validat, s'extrapolarà a les resta de plans.

Aquesta segona iteració la podem segmentar amb els següents passos:

- Captura de senyals
- Conversió de la temperatura de volts a ºC
- Filtratge de les senyals
- Eliminació de l'offset
- Conversió de la velocitat angular de volts a º/s.
- Càlcul del moviment angular

#### Desenvolupament

Per tal d'explicar el desenvolupament d'aquesta segona iteració, seguirem la divisió presentada en l'apartat anterior.

Captura del senyal

Cal aplicar els coneixements de l'iteració anterior per tal de capturar senyal. S'ha de tenir en compte, que ara ja no només capturarem per un PIN sinó que en necessitem dos.

```
word *value;
double mostraT;
double mostraV;
...
```

```
AD1_GetChanValue16(0, value);
mostraT=(double)((*value>>4)*5/4096);

AD1_GetChanValue16(1, value);
mostraV=(double)((*value>>4)*5/4096);
```

En aquest moment, disposem del valor d'una mostra de temperatura en la variable mostraT i d'un valor de velocitat angular en la variable mostraV. A partir d'ara, podem operar amb aquestes variables.

Conversió de la temperatura de volts a $^{o}$C

Aquest segment és molt senzill, només cal aplicar la següent fórmula de conversió:

$$T(^{o}C) = \frac{T(V) - 2.275}{0.009}$$

que a la pràctica, amb el llenguatge de programació C, ho expressem:

```
double mostraTenGraus;
...

mostraTenGraus= (mostraT-2.275)/0.009;
```

Filtratge de les senyals

Filtrarem les dues senyals amb el mateix filtre pas baix. Per tal de crear-lo, utilitzarem l'eina *fdatools* de l'aplicació Matlab. Els paràmetres que cal tenir en compte a l'hora de generar-lo són els següents:

Tipus de filtre: filtre pas baix

Mètode de disseny: FIR

Tipus de finestra: hamming

Nº de coeficients: 101

Frequència de mostreig: 300 Hz

Freqüència de tall: 15 Hz

**Explicació dels paràmetres escollits**

- Número de coeficients: 101
- Freqüència de mostreig: 300 Hz.

Podem expressar el nombre de mostres de retard d'un filtre, segons la següent fórmula:

$$Retard = \frac{N-1}{2} = \frac{ordre}{2}$$

on $N$ és el nombre de coeficients del polinimi, i $ordre$ el seu grau, és a dir, $ordre = N - 1$.

El número de coeficients del filtre sempre ha de ser un nombre senar, donat que ens interessa que el valor resultant de l'expressió anterior sigui un nombre enter, ja que, per exemple, podem esperar el temps que es tarda en capturar 100 mostres (retard en mostres = $\frac{201-1}{2} = 100$), però no el d'obtenir-ne 100'5.

Un cop definit que el retard del filtre és de $\frac{N-1}{2}$ mostres, hem de saber quants segons es tardaran en capturar aquesta quantitat de mostres:

$$temps = n^{o}mostres * Periode = n^{o}mostres * \frac{1}{Frequencia}$$

d'on en podem extreure que:

$$temps = \frac{N-1}{2 * Frequencia}$$

Aquests paràmetres poden prendre infinits valors que facin complir l'equació, però hem de tenir en compte que ens interessa que el temps de retard sigui el mínim possible, que el filtre com més coeficients tingui més bo serà i que hem de mostrejar a una freqüència prou elevada per no perdre informació de la senyal, però tenint en compte no sobremostrejar amb excés, donat que això no ens aporta nova informació. Com que no és possible satisfer-ho tot, s'ha d'arribar a un punt mig. S'ha considerat una bona solució, crear un filtre de 101 coeficients i mostrejar a una freqüència de 300 Hz. Aquesta decisió comporta que el temps de retard és $\frac{1}{6}$ segons.

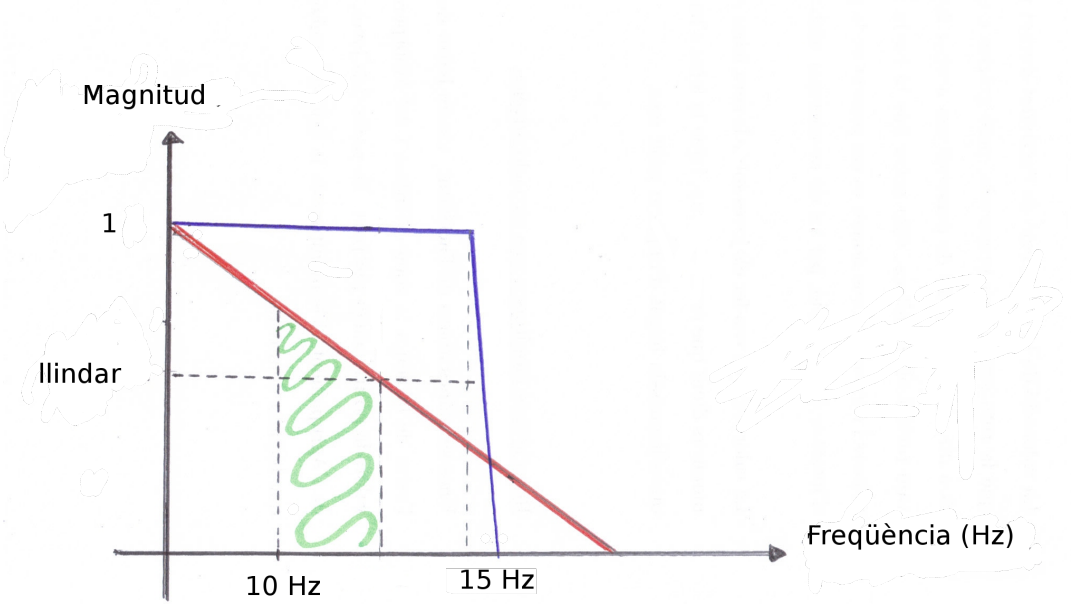


Figura 4.5: Justificació del paràmetre freqüència de tall del filtre pas baix generat

– Freqüència de tall: 15 Hz.

El microcontrolador està limitat per hardware a 10 Hz. Tanmateix, el filtre pas baix que ho implementa és pot considerar dolent, o sigui, no talla als 10 Hz exactes, sinó que considera amples de banda superiors a aquesta quantitat, com a vàlids. Donat que aquest filtre i el que nosaltres disseny estan sèrie, el filtre global que obtenim és el de multiplicar-ne un per l'altre. Si en el nostre filtre tallèssim justament als 10 Hz, estaríem perdent part de la senyal, que no s'havia filtrat correctament en el primer filtre. Per tal d'evitar-ho, augmentem l'ample de banda permès, i ho limitem a 15 Hz.

La interpretació gràfica (figura 4.5) que en podem fer és que tota la zona marcada amb color verd, serien freqüències de la senyal que es perdrien si la freqüència de tall del filtre creat (gràfic blau) fos de 10 Hz, donat que són freqüències superiors a 10 Hz, que un filtre de baixa qualitat (gràfic vermell), com és el que hi ha en el xip, accepta com a vàlides, però que el nostre filtre eliminaria. Així s'explica que s'hagi decidit augmentar aquest valor fins als 15 Hz.

Nota: els filtres de la figura han estat exagerats per tal de facilitar-ne la interpretació.

Mitjançant el programari citat, obtenim els coeficients del filtre, exposats en l'annex B.

El gràfic 4.6 ens mostra el mòdul i la fase del filtre:

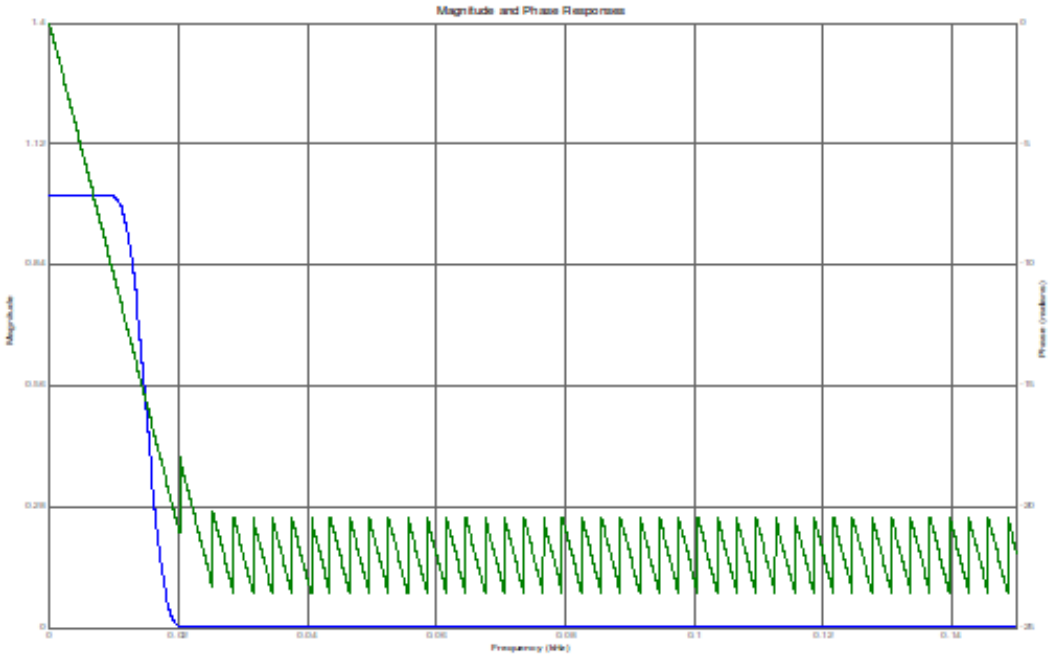


Figura 4.6: Espectre de mòdul i fase del filtre

Un cop dissenyat, l'operació de filtratge és la següent:

```
double mostraCapturada;
double R={0.0};
int n=100;
double h[n]; // vector inicialitzat amb els coeficients del filtre.
...

double filtrarSenyal(mostraCapturada){
    double s=0.0;
    int i;
    for(i=n-1; i>0; i--){
        R[i]=R[i-1];
        s=s+R[i]*h[i];
        }
    R[0]=mostraCapturada;
    return s+R[0]*h[0];
}
```

on:

h: vector que conté els coeficients del filtre.

n: ordre del filtre.

R: registre de desplaçament. Conté un històric de les darreres mostres de la senyal.

mostraCapturada: darrera mostra capturada de la senyal que es vol filtrar.

Aquesta funció ens servirà per filtrar una de les dues senyals. Per fer-ho amb l'altra, el procediment seria el mateix, però caldria substituir el vector R que conté l'històric de mostres d'una senyal, per l'històric de mostres de l'altra.

Matemàticament, el significat de l'algorisme, és la convolució dels coeficients del filtre amb les mostres de la senyal a filtrar.

Eliminació de l'offset

Tal i com s'especifica en l'apartat Punt de partida, l'eliminació de l'offset es tracta d'aplicar la següent expressió, en funció de la temperatura:

$$Offset(T) = p1 * T^9 + p2 * T^8 + p3 * T^7 + p4 * T^6 + p5 * T^5 + p6 * T^4 + p7 * T^3 + p8 * T^2 + p9 * T + p10$$

on els valors de p1, p2,... estan definits al mateix apartat.

En el nostre sistema, es tradueix per:

```
double p[10]={9.436e-019, -2.311e-016, 1.335e-014, 5.797e-013, -5.635e-011,
-5.715e-010, 9.954e-008, 5.424e-007, 0.0002014, 2.5308};

double mostraTenGraus;
...

double eliminarOffset(){
    double s=0;
```

```
    int i;

    for(i=9;i>0; i--){
        T[i]=T[i-1];
        s=s+p[10-i]*T[i];
    }
T[0]=mostraTenGraus;
return s+p[0];
}
```

on el vector T és un històric de les darreres temperatures capturades.

Conversió de la velocitat angular de volts a $^{o}$/s.

Es tracta de fer una conversió d'unitats de la velocitat angular, de volts a $^{o}$/s. Per fer-ho, cal aplicar la següent fórmula:

$$v_{angular}(^{o}/s) = \frac{v_{angular}(V)}{0.00616712 * f_m}$$

i l'equivalent en el nostre sistema és:

```
mostraVAGrausperSegon = mostraVAvolts/(0.0016712*fm)
```

on la freqüència de mostreig és de 300 Hz.

Càlcul del moviment angular

Per tal de calcular el moviment angular total, només ens cal anar acumulant el valor de totes les mostres. Així doncs, tenim que:

```
movAngular=0;
...
```

```
movAngular = movAngular + mostraVAGrausperSegon;
```

**Validació i manteniment**

S'han estudiat els resultats obtinguts, arribant a la conclusió que aquests són prou bons com per continuar avançant amb el desenvolupament del sistema, i extrapolar el que ha estat programat per un pla, als altres dos restants. Com que només ens hem limitat a aplicar l'algorisme fil per randa, l'etapa de manteniment es prorroga fins a la següent iteració.

Capturem dades de velocitat angular i temperatura, sense filtrar les senyals (figura 4.7), tenint en compte que el mòbil estudiat està en repòs i és la temperatura el paràmetre que va variant. En el següent pas en filtrem les senyals (figura 4.8).

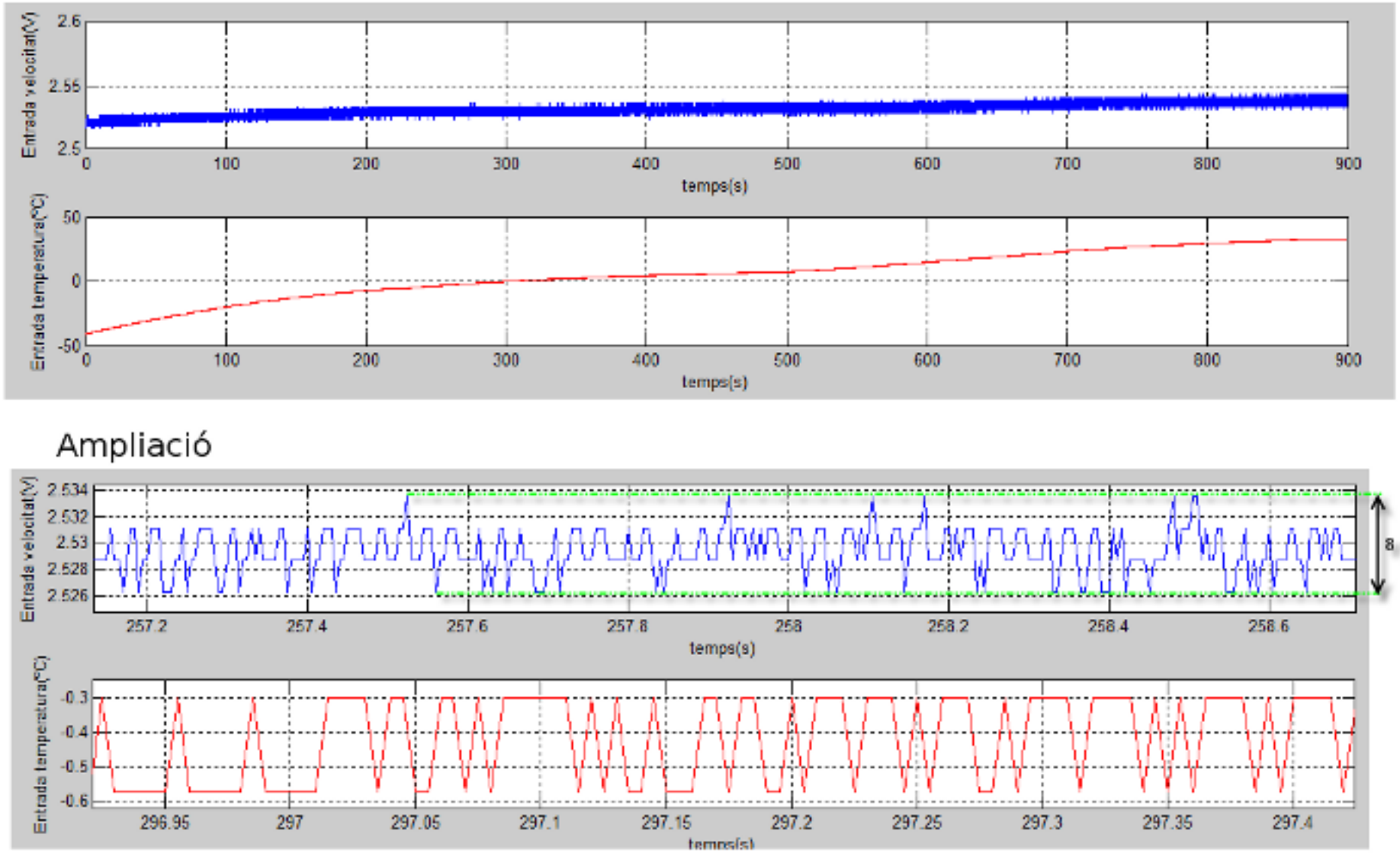


Figura 4.7: Captura (i ampliació) del senyal de velocitat i temperatura

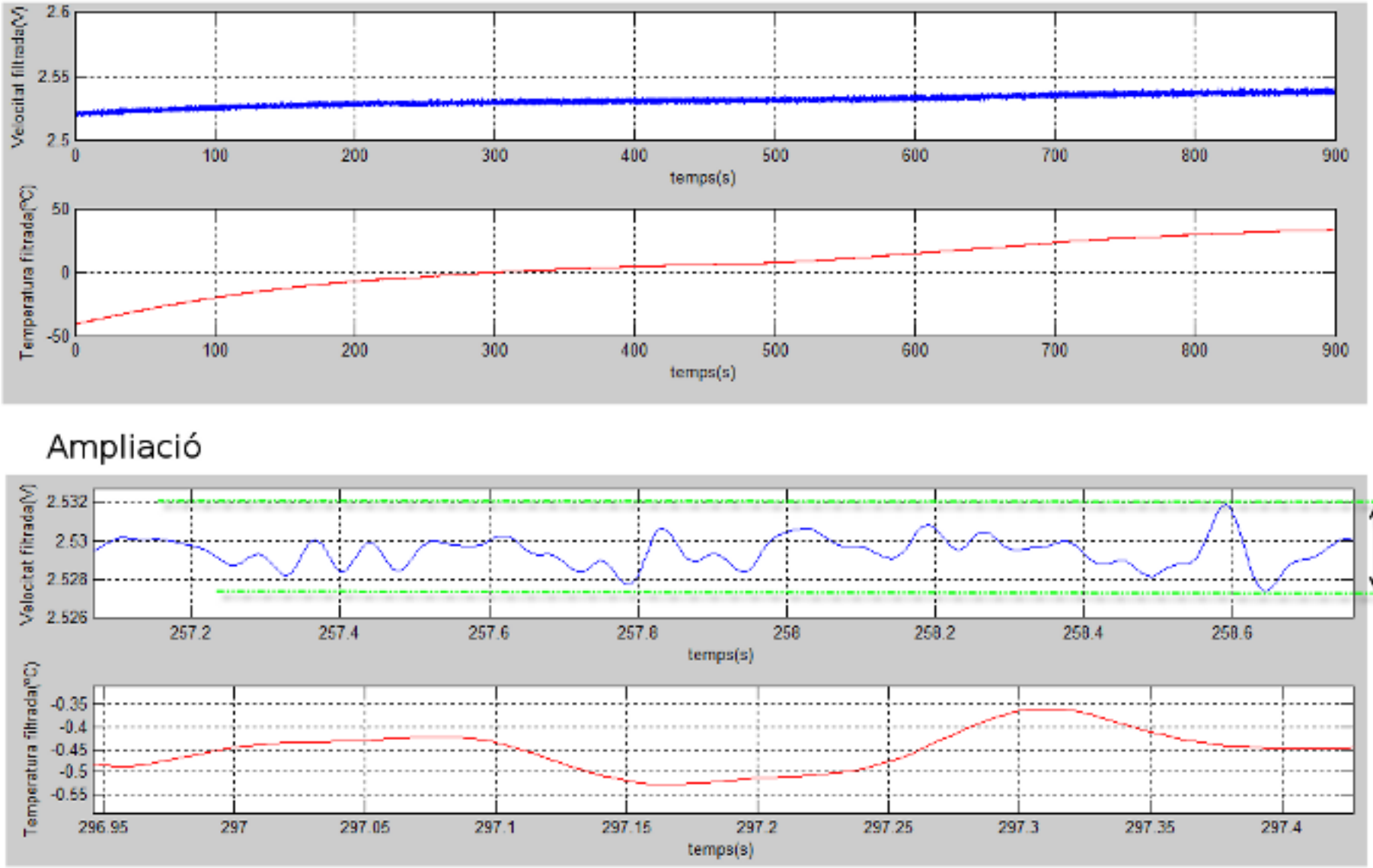


Figura 4.8: Captura (i ampliació) del senyal filtrat de velocitat i temperatura

### 4.3.3 Tercera iteració

#### Anàlisi

Un cop hem desenvolupat l'aplicació per capturar les dades de velocitat angular i temperatura d'un pla, i calcular-ne quin ha estat el seu moviment angular, ja estem en condicions d'extrapolar-ho i aplicar-ho als altres dos plans.

En primer lloc, cal especificar per quin PIN capturarem cada dada:

- AN0. Velocitat angular del pla X.
- AN1. Temperatura del pla X.
- AN3. Velocitat angular del pla Y.
- AN4. Temperatura del pla Y.
- AN5. Velocitat angular del pla Z.
- AN6. Temperatura del pla Z.

#### Desenvolupament

Els algorismes a seguir són exactament els mateixos que en la iteració anterior, aplicant-los ara per triplicat. La variació més destacable respecte l'anterior el trobem en la captura de dades. Hem deixat d'emprar l'operació AD1_GetChanValue16(canal, x) per passar a utilitzar AD1_GetValue16(y).

> on $x$ contenia el valor de la mostra que es capturava pel canal canal, i $y$ serà un vector de 8 posicions (*double*), dels quals només n'emprarem 6, que contindrà les lectures dels 6 canals que utilitzem.

Així doncs, el codi de la nostra aplicació queda de la següent forma:

```
AD1_GetValue16(value);
mostraVeixX=((double)((value[0])>>4))*5/4096;
mostraTeixX=((double)((value[1])>>4))*5/4096;
```

```
mostraVeixY=((double)((value[2])>>4))*5/4096;
mostraTeixY=((double)((value[3])>>4))*5/4096;
mostraVeixZ=((double)((value[4])>>4))*5/4096;
mostraTeixZ=((double)((value[5])>>4))*5/4096;
```

El nombre de variables que s'utilitzen ha augmentat molt. Com ja s'ha justificat en alguna altra ocasió, el motiu d'aquesta situació és facilitar-ne la depuració i trobar les possibles errades que hi puguin haver, ràpidament.

#### Validació i manteniment

No seria factible donar per vàlid el sistema amb la quantitat de recursos que s'utilitzen, i més sabent que la quantitat de memòria que disposem és limitada. Un cop testejats els resultats d'aquesta iteració (fer per triplicat les proves de la fase anterior), ja en podem començar la quarta i definitiva. A diferència de la resta de fases, en aquesta és molt important el manteniment. Ja s'ha anat insistint que era en aquesta iteració, on la depuració del programa era més senzilla i factible de dur a terme. És molt important finalitzar aquesta fase sense errors, donat que en la següent, un cop simplifiquem tot el que sigui possible la quantitat de variables i d'instruccions, en serà m'és complicada la seva detecció.

### 4.3.4 Quarta iteració

#### Anàlisi

Com ja s'ha anat comentat durant tot el procés de desenvolupament de l'aplicació, aquesta darrera fase no afegeix cap mena de funcionalitat, sinó que un cop comprovat que els resultats del sistema en les fases anteriors han estat correctes, arriba el moment d'agilitzar els càlculs i reduir la quantitat de memòria a utilitzar.

És evident que el codi perdrà llegibilitat i que els possibles errors que es produeixin seran força més difícils de detectar, ja que estarà tot més compacte i en serà molt complicada la seva depuració. Tanmateix, aquesta darrera iteració es produeix en un moment en què el programa ja no hauria de contenir cap tipus d'error, és a dir, ja ha passat per tres iteracions en les que s'han hagut de detectar i corregir.

**Desenvolupament**

El codi del sistema està especificat en l'annex C.

A continuació, s'explicaran les característiques més remarcables. Començarem detallant totes les variables que s'empren, tot i que la majoria ja han estat citades en alguna altra iteració, cal destacar que algunes han patit canvis importants: per exemple, passen de ser variables tipus double, a ser vectors d'aquest tipus, o hi ha vectors, que passen de tenir una dimensió, o ser de dos dimensions,.. Tot això és per afavorir la simplificació de variables.

```
Coeficients dels filtres: double h[100];
Històric de velocitats angulars: double R[3][100]
    Eix X: R[0][]
    Eix Y: R[1][]
    Eix Z: R[2][]

Històric de temperatura: double T[3][100]
    Eix X: T[0][]
    Eix Y: T[1][]
    Eix Z: T[2][]

Captura AD_1: word value[8]

Mostres de temperatures: double mostraT[3]
    Eix X: mostraT[0]
    Eix Y: mostraT[1]
    Eix Z: mostraT[2]

Mostres de velocitat angular: double mostraV[3]
    Eix X: mostraV[0]
    Eix Y: mostraV[1]
    Eix Z: mostraV[2]

Moviment angular: double movAngular[3]
    Eix X: movAngular[0]
    Eix Y: movAngular[1]
```

```
    Eix Z: movAngular[2]

Index de pla a utilizar: int x
    Eix X:  x=0
    Eix Y:  x=1
    Eix Z:  x=2
```

Un cop definides les variables globals que s'utilitzen, ens interessa conèixer quin és el comportament del sistema cada cop que s'executa l'event *AD1_OnEnd*.

```
void AD1_OnEnd(void)
{
    capturarSenyals();
    filtrarSenyals();
    for(x=0; x<3; x++)
        movAngular[x]=movAngular[x]+(mostraV[x]-eliminarOffset())/(0.0016712*300);
}
```

Tal i com es pot veure en el codi anterior, podem dividir el sistema en tres fragments diferenciats:

Captura de les senyals

Filtratge de les senyals

Càlcul

Captura de les senyals

Ja s'ha explicat com es duia a terme la captura de les senyals, tanmateix és important afegir, que en el moment de llegir les mostres de temperatura, ja s'opera amb elles, fent la conversió a $^{o}$/s, ja que per temes de rendiment, s'ha comprovat que és més ràpid fer-ho amb una única instrucció que no pas amb dos de separades. Per tant, el codi queda de la següent forma:

```
AD1_GetValue16(value);
mostraV[0]=((double)((value[0])>>4))*5/4096;
mostraT[0]=(((double)((value[1])>>4))-2.275)/(0.009*4096/5); // en graus
mostraV[1]=(double)((value[2])>>4)*5/4096;
mostraT[1]=(((double)((value[3])>>4))-2.275)/(0.009*4096/5);
mostraV[2]=(double)((value[4])>>4)*5/4096;
mostraT[2]=(((double)((value[5])>>4))-2.275)/(0.009*4096/5);
```

on també ens podem fixar que no s'utilitza una variable per a cada eix, sinó que fem ús de vectors.

Filtratge de les senyals

La idea de l'algorisme és la mateixa que s'exposava en la segona iteració. Però en aquesta última fase s'han agrupat els dos filtres sota una única funció, que enlloc de retornar una quantitat (double), modifica directament el valor de les mostres:

```
void filtrarSenyal() {
    float sv=0.0;
    float st=0.0; //sumes parcials
    int i;
    for(i=n-1; i>0; i--){
        R[x][i]=R[x][i-1];
        T[x][i]=T[x][i-1];
        sv=sv+R[x][i]*h[i];
        st=st+T[x][i]*h[i];
    }
    R[x][0]=mostraT[x];
    T[x][0]=mostraV[x];
    mostraT[x]=st+T[x][0]*h[0];
    mostraV[x]=sv+R[x][0]*h[0];
}
```

El rerefons de l'algorisme és exactament el mateix que en la segona iteració, és a dir, la convolució dels coeficients del filtre amb les mostres de la senyal a filtrar.

Càlculs

El càlcul que queda resumit amb un sola instrucció amb crides a funcions, inclou:

- Eliminació de l'offset
- Conversió a º/s
- Integral per calcular el moviment angular del pla.

No entrarem a detallar quin és el funcionament del codi que implementen aquestes operacions perquè ja s'ha fet en les iteracions anteriors i l'algorisme a seguir no ha canviat. Només insistir en el fet que s'ha introduït la millora de reduir el nombre de variables. Per tota la informació que s'havia d'emmagatzemar de cada pla, s'ha passat d'usar tres variables diferents, a agrupar-ho amb vectors. D'aquí la necessitat de crear una variable x que indica en quin pla estem treballant.

**Validació i manteniment**

Com ja s'ha comentat en l'anàlisi d'aquesta iteració, no és la millor fase per dur a terme un manteniment del sistema. És important que tots els errors de l'aplicació es detectin en la tercera iteració, on la gran quantitat de variables que disposem ens permet ubicar-lo molt més ràpidament, dirigint-nos directament al focus del problema. S'han fet les proves necessàries per comprovar el correcte funcionament del sistema, tenint en compte que l'algorisme que es desenvolupa és exactament el mateix que en la iteració anterior.

## 4.4 Temporització

Davant de qualsevol projecte és molt important saber-lo divir en tasques més simples, i quantificar el temps necessari per tal de desenvolupar-les. Això no implica que no es pugi treballar al mateix temps amb més d'una etapa. És vital fer una previsió, tot l'exhaustiva que sigui possible, abans d'iniciar-lo, per saber quina és la dedicació que requereix el treball i tenir-la com a pauta a seguir, evitant estancar-se en punts molt concrets. També és molt important que a mesura que es vagin finalitzant les tasques que composen el projecte, es vagi revisant aquesta previsió realitzada, i valorar si es necessari aplicar-hi algun canvi.

La següent taula resumeix les principals tasques en què s'ha dividit el desenvolupament del sistema, amb el temps previst inicialment i la revisió d'aquest a la seva finalització.

| Tasca | Temps previst | Temps real |
|---|---|---|
| Lectura de documentació inicial | 2 setmanes | 2 setmanes |
| Recerca i tria dels dispositius a emprar | 1 setmana | 2 setmanes |
| Descoberta de l'entorn *CodeWarrior* | 2 setmanes | 2 setmanes |
| Captura de dades | 2 setmanes | 2 setmanes |
| Implementació de l'algorisme | 4 setmanes | 4.5 setmanes |
| Proves de l'algorisme | 1 setmana | 1 setmana |
| Millores de rendiment | 2 setmanes | 2 setmanes |
| **TOTAL** | **14 setmanes** | **15.5 setmanes** |

Taula 4.1: Temporització del projecte

Observació: quantifiquem el temps en setmanes de treball. Per fer una aproximació en hores, podem considerar 1 setmana com unes 25 hores.

Cal remarcar que la majoria de temps assignats prèviament a cada tasca és el que s'ha empleat després realment. Això no vol dir que la previsió feta inicialment fos perfecta, sinó que ens hem cenyit a aquesta com a guia, donat que sinó es fa complicat donar una tasca per finalitzada, ja que sempre es pot aprofundir més. Podem concloure que el fet d'haver realitzat aquesta temporització ens ha ajudat molt en el desenvolupament del sistema.

## 4.5 Materials i dispositius emprats

Per tal de poder desenvolupar el sistema, s'ha utilitzat el següent material o dispositius:

- Ordinador amb el sistema operatiu Microsoft Windows XP. Software instal·lat:
  - Entorn de programació de microcontroladors Freescale: CodeWarrior
  - HyperTerminal
  - Microsoft Office Excel
  - Matlab (eina *fdatools*)
- Sensor ADXRS610
- Board amb el xip MCF5213
- Oscil·loscopi
- Generador de funcions i font d'alimentació

# Capítol 5

# Conclusions i treball futur

Per tal de poder extreure conclusions del treball realitzat, cal valorar si s'han assolit els objectius marcats. L'objectiu principal del projecte el podem desglossar en dues parts:

- Captura de senyals
- Implementació de l'algorisme de càlcul de moviment angular en cada pla

En termes generals, podem concloure que s'han assolit aquests dos ítems i que per tant s'han satisfet les aspectatives que es tenien al moment d'iniciar el treball. Tanmateix, això no implica que la feina ja estigui acabada. Hem de recordar que aquest projecte finalitza amb la informació del moviment angular de cada pla emmagatzemada en variables. Això ens ha servit en tot el procés de càlcul, però per tal que aquesta informació pugui ser utilitzada per altres sistemes (englobats dins del sistema de major envergadura de què forma part aquest projecte), necessitem poder enviar aquesta informació. Una bona solució podria ser l'ús d'un mòdul I2C.

Per satisfer l'objectiu general del projecte, en calia assolir d'altres de més específics:

- Desenvolupar aplicacions emprant el llenguatge C, tenint en compte que la quantitat de memòria que disposem està limitada pels components escollits i que el seu rendiment ha de ésser el més elevat possible.
- Programar el microcontrolador MCF5213 mitjançant l'eina de desenvolupament Code Warrior 7.1.

- Conèixer els diferents mòduls que composen el xip en qüestió, distingint quins ens poden ésser útils en cada moment, i saber-los utilitzar de forma correcta.

Si s'han assolit les fites de capturar senyal i implementar l'algorisme, ha estat perquè abans s'han assolit aquests objectius més específics. Per tant, podem concloure que s'han complert tots els objectius proposats.

La tasca més important que manca per fer, en futurs treballs, és la interconnexió amb els altres sistemes que en conformen el projecte global.

# Apèndix A

# Especificació dels PINs del microcontrolador MCF5213

A continuació es detalla el funcionament de cada pin del microcontrolador MCF5213.

Els camps més rellavants pel desenvolupament del projecte són:

- Grup del qual forma part el PIN
- Funció principal
- Funció secundària
- Posició del PIN en el xip.

| Pin Group | Primary Function | Secondary Function | Tertiary Function | Quaternary Function | Drive Strength/ Control[1] | Slew Rate/ Control[1] | Pull-up / Pull-down[2] | Pin on 100 LQFP | Pin on 81 MAPBGA | Pin on 64 LQFP/QFN | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ADC | AN7 | — | — | GPIO | Low | FAST | — | 51 | H9 | 33 | |
| | AN6 | — | — | GPIO | Low | FAST | — | 52 | G9 | 34 | |
| | AN5 | — | — | GPIO | Low | FAST | — | 53 | G8 | 35 | |
| | AN4 | — | — | GPIO | Low | FAST | — | 54 | F9 | 36 | |
| | AN3 | — | — | GPIO | Low | FAST | — | 46 | G7 | 28 | |
| | AN2 | — | — | GPIO | Low | FAST | — | 45 | G6 | 27 | |
| | AN1 | — | — | GPIO | Low | FAST | — | 44 | H6 | 26 | |
| | AN0 | — | — | GPIO | Low | FAST | — | 43 | J6 | 25 | |
| | SYNCA | — | — | — | N/A | N/A | — | — | — | — | No Primary |
| | SYNCB | — | — | — | N/A | N/A | — | — | — | — | No Primary |
| | VDDA | — | — | — | N/A | N/A | — | 50 | H8 | 32 | |
| | VSSA | — | — | — | N/A | N/A | — | 47 | H7, J9 | 29 | |
| | VRH | — | — | — | N/A | N/A | — | 49 | J8 | 31 | |
| | VRL | — | — | — | N/A | N/A | — | 48 | J7 | 30 | |
| Clock Generation | EXTAL | — | — | — | N/A | N/A | — | 73 | B9 | 47 | |
| | XTAL | — | — | — | N/A | N/A | — | 72 | C9 | 46 | |
| | VDDPLL | — | — | — | N/A | N/A | — | 74 | B8 | 48 | |
| | VSSPLL | — | — | — | N/A | N/A | — | 71 | C8 | 45 | |
| Debug Data | ALLPST | — | — | — | High | FAST | — | 86 | A6 | 55 | |
| | DDATA[3:0] | — | — | GPIO | High | FAST | — | 84,83,78,77 | — | — | |
| | PST[3:0] | — | — | GPIO | High | FAST | — | 70,69,66,65 | — | — | |
| $I^2C$ | SCL | CANTX[3] | UTXD2 | GPIO | PDSR[0] | PSRR[0] | pull-up[4] | 10 | E1 | 8 | |
| | SDA | CANRX[3] | URXD2 | GPIO | PDSR[0] | PSRR[0] | pull-up[4] | 11 | E2 | 9 | |

Taula A.1: Especificació dels PINs del MCF5213

| Pin Group | Primary Function | Secondary Function | Tertiary Function | Quaternary Function | Drive Strength/ Control[1] | Slew Rate/ Control[1] | Pull-up / Pull-down[2] | Pin on 100 LQFP | Pin on 81 MAPBGA | Pin on 64 LQFP/QFN | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Interrupts | $\overline{\text{IRQ7}}$ | — | — | GPIO | Low | FAST | pull-up | 95 | C4 | 58 | |
| | $\overline{\text{IRQ6}}$ | — | — | GPIO | Low | FAST | pull-up | 94 | B4 | — | |
| | $\overline{\text{IRQ5}}$ | — | — | GPIO | Low | FAST | pull-up | 91 | A4 | — | |
| | $\overline{\text{IRQ4}}$ | — | — | GPIO | Low | FAST | pull-up | 90 | C5 | 57 | |
| | $\overline{\text{IRQ3}}$ | — | — | GPIO | Low | FAST | pull-up | 89 | A5 | — | |
| | $\overline{\text{IRQ2}}$ | — | — | GPIO | Low | FAST | pull-up | 88 | B5 | — | |
| | $\overline{\text{IRQ1}}$ | SYNCA | PWM1 | GPIO | High | FAST | pull-up[4] | 87 | C6 | 56 | |
| JTAG/BDM | JTAG_EN | — | — | — | N/A | N/A | pull-down | 26 | J2 | 17 | |
| | TCLK/ PSTCLK | CLKOUT | — | — | High | FAST | pull-up[5] | 64 | C7 | 44 | |
| | TDI/DSI | — | — | — | N/A | N/A | pull-up[5] | 79 | B7 | 50 | |
| | TDO/DSO | — | — | — | High | FAST | — | 80 | A7 | 51 | |
| | TMS /$\overline{\text{BKPT}}$ | — | — | — | N/A | N/A | pull-up[5] | 76 | A8 | 49 | |
| | $\overline{\text{TRST}}$ /DSCLK | — | — | — | N/A | N/A | pull-up[5] | 85 | B6 | 54 | |
| Mode Selection[6] | CLKMOD0 | — | — | — | N/A | N/A | pull-down[6] | 40 | G5 | 24 | |
| | CLKMOD1 | — | — | — | N/A | N/A | pull-down[6] | 39 | H5 | — | |
| | $\overline{\text{RCON}}$/ EZPCS | — | — | — | N/A | N/A | pull-up | 21 | G3 | 16 | |
| PWM | PWM7 | — | — | GPIO | PDSR[31] | PSRR[31] | — | 63 | D7 | — | |
| | PWM5 | — | — | GPIO | PDSR[30] | PSRR[30] | — | 60 | E8 | — | |
| | PWM3 | — | — | GPIO | PDSR[29] | PSRR[29] | — | 33 | J4 | — | |
| | PWM1 | — | — | GPIO | PDSR[28] | PSRR[28] | — | 38 | J5 | — | |

Taula A.2: Especificació dels PINs del MCF5213 (continuació)

| Pin Group | Primary Function | Secondary Function | Tertiary Function | Quaternary Function | Drive Strength/ Control[1] | Slew Rate/ Control[1] | Pull-up / Pull-down[2] | Pin on 100 LQFP | Pin on 81 MAPBGA | Pin on 64 LQFP/QFN | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|
| QSPI | QSPI_DIN/ EZPD | CANRX[3] | URXD1 | GPIO | PDSR[2] | PSRR[2] | — | 16 | F3 | 12 | |
| | QSPI_DOUT /EZPQ | CANTX[3] | UTXD1 | GPIO | PDSR[1] | PSRR[1] | — | 17 | G1 | 13 | |
| | QSPI_CLK/ EZPCK | SCL | $\overline{\text{URTS1}}$ | GPIO | PDSR[3] | PSRR[3] | pull-up[7] | 18 | G2 | 14 | |
| | QSPI_CS3 | SYNCA | SYNCB | GPIO | PDSR[7] | PSRR[7] | —pull-up/pull-down | 12 | F1 | — | |
| | QSPI_CS2 | — | — | GPIO | PDSR[6] | PSRR[6] | — | 13 | F2 | — | |
| | QSPI_CS1 | — | — | GPIO | PDSR[5] | PSRR[5] | — | 19 | H2 | — | |
| | QSPI_CS0 | SDA | $\overline{\text{UCTS1}}$ | GPIO | PDSR[4] | PSRR[4] | pull-up[7] | 20 | H1 | 15 | |
| Reset[8] | $\overline{\text{RSTI}}$ | — | — | — | N/A | N/A | pull-up[8] | 96 | A3 | 59 | |
| | $\overline{\text{RSTO}}$ | — | — | — | high | FAST | — | 97 | B3 | 60 | |
| Test | TEST | — | — | — | N/A | N/A | pull-down | 5 | C2 | 3 | |
| Timers, 16-bit | GPT3 | — | PWM7 | GPIO | PDSR[23] | PSRR[23] | pull-up[9] | 62 | D8 | 43 | |
| | GPT2 | — | PWM5 | GPIO | PDSR[22] | PSRR[22] | pull-up[9] | 61 | D9 | 42 | |
| | GPT1 | — | PWM3 | GPIO | PDSR[21] | PSRR[21] | pull-up[9] | 59 | E9 | 41 | |
| | GPT0 | — | PWM1 | GPIO | PDSR[20] | PSRR[20] | pull-up[9] | 58 | F7 | 40 | |
| Timers, 32-bit | DTIN3 | DTOUT3 | PWM6 | GPIO | PDSR[19] | PSRR[19] | — | 32 | H3 | 19 | |
| | DTIN2 | DTOUT2 | PWM4 | GPIO | PDSR[18] | PSRR[18] | — | 31 | J3 | 18 | |
| | DTIN1 | DTOUT1 | PWM2 | GPIO | PDSR[17] | PSRR[17] | — | 37 | G4 | 23 | |
| | DTIN0 | DTOUT0 | PWM0 | GPIO | PDSR[16] | PSRR[16] | — | 36 | H4 | 22 | |
| UART 0 | $\overline{\text{UCTS0}}$ | CANRX | — | GPIO | PDSR[11] | PSRR[11] | — | 6 | C1 | 4 | |
| | $\overline{\text{URTS0}}$ | CANTX | — | GPIO | PDSR[10] | PSRR[10] | — | 9 | D3 | 7 | |
| | URXD0 | — | — | GPIO | PDSR[9] | PSRR[9] | — | 7 | D1 | 5 | |
| | UTXD0 | — | — | GPIO | PDSR[8] | PSRR[8] | — | 8 | D2 | 6 | |

Taula A.3: Especificació dels PINs del MCF5213 (continuació)

| Pin Group | Primary Function | Secondary Function | Tertiary Function | Quaternary Function | Drive Strength/ Control[1] | Slew Rate/ Control[1] | Pull-up / Pull-down[2] | Pin on 100 LQFP | Pin on 81 MAPBGA | Pin on 64 LQFP/QFN | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|
| UART 1 | $\overline{\text{UCTS1}}$ | SYNCA | URXD2 | GPIO | PDSR[15] | PSRR[15] | — | 98 | C3 | 61 | |
| | $\overline{\text{URTS1}}$ | SYNCB | UTXD2 | GPIO | PDSR[14] | PSRR[14] | — | 4 | B1 | 2 | |
| | URXD1 | — | — | GPIO | PDSR[13] | PSRR[13] | — | 100 | B2 | 63 | |
| | UTXD1 | — | — | GPIO | PDSR[12] | PSRR[12] | — | 99 | A2 | 62 | |
| UART 2 | $\overline{\text{UCTS2}}$ | — | — | GPIO | PDSR[27] | PSRR[27] | — | 27 | — | — | |
| | $\overline{\text{URTS2}}$ | — | — | GPIO | PDSR[26] | PSRR[26] | — | 30 | — | — | |
| | URXD2 | — | — | GPIO | PDSR[25] | PSRR[25] | — | 28 | — | — | |
| | UTXD2 | — | — | GPIO | PDSR[24] | PSRR[24] | — | 29 | — | — | |
| FlexCAN | CANRX | | | | N/A | N/A | — | — | — | — | See Note[3,10] |
| | CANTX | | | | N/A | N/A | — | — | — | — | See Note[3,10] |
| VSTBY | VSTBY | — | — | — | N/A | N/A | — | 55 | F8 | 37 | |
| VDD | VDD | — | — | — | N/A | N/A | — | 1,2,14,22, 23,34,41, 57,68,81,93 | D5,E3–E7, F5 | 1,10,20,39, 52 | |
| VSS | VSS | — | — | — | N/A | N/A | — | 3,15,24,25, 35,42,56, 67,75,82,92 | A1,A9,D4, D6,F4,F6, J1 | 11,21,38, 53,64 | |

Taula A.4: Especificació dels PINs del MCF5213 (continuació)

# Apèndix B

# Coeficients dels filtres pas baix

A continuació es detallen els coeficients del filtre utilitzat. Han estat generats mitjançant l'eina *fdatool* de Matlab.

| | |
|---|---|
| 0.000000000000000 | 0.000000000000000 |
| 0.000162415058636 | 0.001443573226873 |
| 0.000325968268401 | 0.003032352398774 |
| 0.000482971729359 | 0.004600481945529 |
| 0.000621596572603 | 0.005951177284044 |
| 0.000725138232653 | 0.006875493541570 |
| 0.000772670665657 | 0.007175772391151 |
| 0.000741290161892 | 0.006691588846216 |
| 0.000609845139709 | 0.005325446278758 |
| 0.000363725759805 | 0.003065171136843 |
| -0.000000000000000 | -0.000000000000000 |
| -0.000468016748823 | -0.003672245306467 |
| -0.001007740722249 | -0.007648752410270 |
| -0.001566742471065 | -0.011536912553795 |
| -0.002075260639149 | -0.014880733300811 |
| -0.002452066412305 | -0.017196823509853 |
| -0.002613453351758 | -0.018017125983100 |
| -0.002484607350841 | -0.016934621889451 |
| -0.002012129434797 | -0.013647628154408 |
| -0.001176114730005 | -0.007998142204448 |

| | |
|---|---|
| 0.000000000000000 | -0.000000000000000 |
| 0.010146606811222 | 0.003065171136843 |
| 0.022056523678821 | 0.005325446278758 |
| 0.035177725575843 | 0.006691588846216 |
| 0.048825272913973 | 0.007175772391151 |
| 0.062228691321826 | 0.006875493541570 |
| 0.074588926108252 | 0.005951177284044 |
| 0.085139961570803 | 0.004600481945529 |
| 0.093209604207554 | 0.003032352398774 |
| 0.098273879579863 | 0.001443573226873 |
| 0.100000000000000 | 0.000000000000000 |
| 0.098273879579863 | -0.001176114730005 |
| 0.093209604207554 | -0.002012129434797 |
| 0.085139961570803 | -0.002484607350841 |
| 0.074588926108252 | -0.002613453351758 |
| 0.062228691321826 | -0.002452066412305 |
| 0.048825272913973 | -0.002075260639149 |
| 0.035177725575843 | -0.001566742471065 |
| 0.022056523678821 | -0.001007740722249 |
| 0.010146606811222 | -0.000468016748823 |
| 0.000000000000000 | -0.000000000000000 |
| -0.007998142204448 | 0.000363725759805 |
| -0.013647628154408 | 0.000609845139709 |
| -0.016934621889451 | 0.000741290161892 |
| -0.018017125983100 | 0.000772670665657 |
| -0.017196823509853 | 0.000725138232653 |
| -0.014880733300811 | 0.000621596572603 |
| -0.011536912553795 | 0.000482971729359 |
| -0.007648752410270 | 0.000325968268401 |
| -0.003672245306467 | 0.000162415058636 |
| | 0.000000000000000 |

# Apèndix C

# Codi font del programa

Tal i com s'ha esmentant en l'apartat de desenvolupament del sistema, el fitxer principal (main.c) només conté una instrucció, per tal d'iniciar el conversor analògic-digital:

```
AD1_Start();
```

A continuació detallem el codi font generat en el document que gestiona els events (Events.c).

```
int n=100;
double p[10]={9.436e-019, -2.311e-016, 1.335e-014, 5.797e-013, -5.635e-011,
-5.715e-010, 9.954e-008, 5.424e-007, 0.0002014, 2.5308};

double h[100];
double R[3][100]={0.0};
double T[3][100]={0.0};

word value[8];

double mostraT[3]={0.0};
double mostraV[3]={0.0};

int x=0;
```

```
double movAngular[3]={0.0};


double filtrarSenyal(void);
void eliminarOffset(void);
void capturarSenyals(void);



void AD1_OnEnd(void){

    capturarSenyals();
    filtrarSenyals();
    for(x=0; x<3; x++)
    movAngular[x]=movAngular[x]+(mostraV[x]-eliminarOffset())/(0.0016712*300);
        // Freqüència de mostreig = 300 Hz
}


void capturarSenyals(void){

    AD1_GetValue16(value);
    mostraV[0]=((double)((value[0])>>4))*5/4096;
    mostraV[1]=(double)((value[2])>>4)*5/4096;
    mostraV[2]=(double)((value[4])>>4)*5/4096;

    mostraT[0]= ((((double)(value[1]>>4))*5/4096)-2.275)/0.009;
    mostraT[1]= ((((double)(value[3]>>4))*5/4096)-2.275)/0.009;
    mostraT[2]= ((((double)(value[5]>>4))*5/4096)-2.275)/0.009;
}


void filtrarSenyal() {
    float sv=0.0;
    float st=0.0;
    int i;
    for(i=n-1; i>0; i--){
```

```
        R[x][i]=R[x][i-1];
        T[x][i]=T[x][i-1];
        sv=sv+R[x][i]*h[i];
        st=st+T[x][i]*h[i];
    }
    R[x][0]=mostraT[x];
    T[x][0]=mostraV[x];
    mostraT[x]=st+T[x][0]*h[0];
    mostraV[x]=sv+R[x][0]*h[0];
}


double eliminarOffset(){

    float s=0;
    int i;
    for(i=9;i>0; i--){
        T[x][i]=T[x][i-1];
        s=s+p[10-i]*T[x][i];
    }
    T[x][0]=mostraT[x];
    return s+p[0];
}
```

Nota: Per motius de practicitat i llegibilitat del codi font, el vector $h$ que conté els coeficients dels filtres detallats en l'annex B, no ha estat inicialitzat.

# Bibliografia

[1] Aplicació d'estratègies de processament de senyals variants amb la velocitat angular del dispositiu sensor ADXRS610. Treball final de carrera d'en Robert Rocamora Graell dirigit per en Francisco Clariá Sancho. Maig de 2009.

[2] Processat de senyals d'acceleració i metodologia d'ajust de zero. Treball final de carrera de n'Albert Agraz Sánchez dirigit per en Francisco Clariá Sancho. Gener de 2010.

[3] MCF5213 ColdFire Integrated Microcontroller Reference Manual. Freescale Semiconductor. Març de 2007.

[4] ADXRS610. Analog devices. Abril de 2007.

[5] Ingeniería del software: un enfoque práctico. Mc. Graw Hill. Any 2001. Cinquena edició.

[6] Ingeniería del software. Addison-Wesley. Any 2002. Sisena edició.